AGARD-R-787

AGARD-R-787

①

AGARD

ADVISORY GROUP FOR AEROSPACE RESEARCH & DEVELOPMENT

7 RUE ANCELLE 92200 NEUILLY SUR SEINE FRANCE

AGARD REPORT 787

Special Course on Unstructured Grid Methods for Advection Dominated Flows

(Les Méthodes Utilisant un Maillage non Structuré pour l'Etude des Ecoulements Caracterisés par l'Advection)

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented in an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 2nd—6th March 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

NORTH ATLANTIC TREATY ORGANIZATION

Published May 1992

Distribution and Availability on Back Cover

AGARD

ADVISORY GROUP FOR AEROSPACE RESEARCH & DEVELOPMENT

7 RUE ANCELLE 92200 NEUILLY SUR SEINE FRANCE

AGARD REPORT 787

Special Course on Unstructured Grid Methods for Advection Dominated Flows

(Les Méthodes Utilisant un Maillage non Structuré pour l'Etude des Ecoulements Caracterisés par l'Advection)

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented in an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 2nd—6th March 1992 and at the NASA Ames Research Center, Moffett Field, California, United States 28th September--2nd October 1992.

North Atlantic Treaty Organization

Organisation du Traité de l'Atlantique Nord

The Mission of AGARD

According to its Charter, the mission of AGARD is to bring together the leading personalities of the NATO nations in the fields of science and technology relating to aerospace for the following purposes:

- Recommending effective ways for the member nations to use their research and development capabilities for the common benefit of the NATO community;
- Providing scientific and technical advice and assistance to the Military Committee in the field of aerospace research and development (with particular regard to its military application);
- Continuously stimulating advances in the aerospace sciences relevant to strengthening the common defence posture;
- Improving the co-operation among member nations in aerospace research and development;
- Exchange of scientific and technical information;
- Providing assistance to member nations for the purpose of increasing their scientific and technical potential;
- Rendering scientific and technical assistance, as requested, to other NATO bodies and to member nations in connection with research and development problems in the aerospace field.

The highest authority within AGARD is the National Delegates Board consisting of officially appointed senior representatives from each member nation. The mission of AGARD is carried out through the Panels which are composed of experts appointed by the National Delegates, the Consultant and Exchange Programme and the Aerospace Applications Studies Programme. The results of AGARD work are reported to the member nations and the NATO Authorities through the AGARD series of publications of which this is one.

Participation in AGARD activities is by invitation only and is normally limited to citizens of the NATO nations.

The content of this publication has been reproduced
directly from material supplied by AGARD or the authors.

Published May 1992

ISBN 92-835-0671-5

SPS

Printed by Specialised Printing Services Limited
40 Chigwell Lane, Loughton, Essex IG10 3TZ

Aerodynamics of Combat Aircraft Controls and of Ground Effects
AGARD CP-465, April 1990

Computational Methods for Aerodynamic Design (Inverse) and Optimization
AGARD-CP-463, March 1990

Applications of Mesh Generation to Complex 3-D Configurations
AGARD CP-464, March 1990

Fluid Dynamics of Three-Dimensional Turbulent Shear Flows and Transition
AGARD CP-438, April 1989

Validation of Computational Fluid Dynamics
AGARD CP-437, December 1988

Aerodynamic Data Accuracy and Quality: Requirements and Capabilities in Wind Tunnel Testing
AGARD CP-429, July 1988

Aerodynamics of Hypersonic Lifting Vehicles
AGARD CP-428, November 1987

Aerodynamic and Related Hydrodynamic Studies Using Water Facilities
AGARD CP-413, June 1987

Applications of Computational Fluid Dynamics in Aeronautics
AGARD CP-412, November 1986

Store Airframe Aerodynamics
AGARD CP-389, August 1986

Unsteady Aerodynamics — Fundamentals and Applications to Aircraft Dynamics
AGARD CP-386, November 1985

Aerodynamics and Acoustics of Propellers
AGARD CP-366, February 1985

Improvement of Aerodynamic Performance through Boundary Layer Control and High Lift Systems
AGARD CP-365, August 1984

Wind Tunnels and Testing Techniques
AGARD CP-348, February 1984

Aerodynamics of Vortical Type Flows in Three Dimensions
AGARD CP-342, July 1983

Missile Aerodynamics
AGARD CP-336, February 1983

Prediction of Aerodynamic Loads on Rotorcraft
AGARD CP-334, September 1982

Wall Interference in Wind Tunnels
AGARD CP-335, September 1982

Fluid Dynamics of Jets with Applications to V/STOL
AGARD CP-308, January 1982

Aerodynamics of Power Plant Installation
AGARD CP-301, September 1981

Computation of Viscous-Inviscid Interactions
AGARD CP-291, February 1981

Subsonic/Transonic Configuration Aerodynamics
AGARD CP-285, September 1980

Turbulent Boundary Layers Experiments, Theory and Modelling
AGARD CP-271, January 1980

Aerodynamic Characteristics of Controls
AGARD CP-262, September 1979

High Angle of Attack Aerodynamics
AGARD CP-247, January 1979

Foreword

The objective of the course and the course notes is to present the state-of-the-art, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Topics and methods covered include:

— Least Squares Galerkin and Streamline Diffusion Finite Element Methods

— Finite Volume Methods and Higher Order Polynomial Reconstruction

— Essentially Non Oscillatory Schemes for Unstructured Grids

— Multidimensional Upwind Schemes on Triangles and Tetrahedra

— Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle

— Turbulence Modelling on Unstructured Grids

— Error Estimators and Solution Adaptivity

— Parallel Computing on Unstructured Grid

— Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, going from incompressible free surface problems transonic aerodynamics and hypersonic reentry flows.

Getting these notes prepared in time was a difficult task. Only those who have been in this situation realize the time and work it needs to write such lectures. We wish to congratulate all lecturers, who in between their busy professional activities have found, or better made, the time to write detailed and high quality contributions.

It is our opinion that we have here a reference work which will be used by a whole generation of PhDs and other researchers who wish to jump into the absorbing subject of unstructured grid methods in CFD.

We are convinced that the lecture series for which these notes have been made will be a great success, and we thank both our institutions, the von Kármán Institute and the NASA Ames Research Center for hosting the course. We thank AGARD and the Fluid Dynamics Panel, its past Chairman Jim McCroskey, Executive Winston Goodrich and Secretary Anne-Marie Rivault for their encouragement and professional support.

Herman Deconinck and Tim Barth
Course Directors

Avant-Propos

L'objet de ce cours et du support de cours est de présenter l'état de l'art, ainsi que certains développements récents dans le domaine des méthodes en maillages non-structurés pour le calcul d'écoulements compressibles et non-compressibles à grand nombre de Reynolds et d'autres sujets connexes. Les questions et les méthodes couvertes sont les suivantes:

— méthodes des éléments finis moindres carrés Galerkin et à diffusion longitudinale

— méthodes de volumes finis et reconstruction polynomiale d'ordre élevé

— méthodes essentiellement non-oscillatoires pour maillages non-structurés

— schémas multidimensionnels de discrétisation décentrée sur triangles et tétraèdres

— méthodes de génération de maillages pour grilles non structurées selon la méthode frontale et le principe de Delaunay

— modélisation de la turbulence sur maillages non-structurés

— estimateurs d'erreur et algorithmes pour maillages adaptatifs

— calcul en parallèle sur maillages non-structurés

— post-traitement des bases de données sur maillages non structurés pour la visualisation des écoulements.

Un large éventail d'applications est présenté, allant des problèmes de surfaces libres non-compressibles à l'aérodynamique transsonique et les écoulements hypersoniques rentrants.

La mise en forme de ce support de cours dans les délais impartis s'est avérée difficile. Seuls ceux qui ont vécu une telle situation seront conscients du temps et des efforts qu'il faut consacrer à la rédaction de tels cours. Nous tenons donc, à féliciter tous nos conférenciers, qui, malgré des emplois de temps très chargés, ont trouvé, ou plutôt créé, le temps nécessaire pour écrire des textes détaillés de grande qualité.

A notre avis, il s'agit d'un véritable ouvrage de référence qui trouvera un large accueil auprès de toute une génération de diplômés troisième cycle et d'autres chercheurs souhaitant découvrir le sujet passionnant des méthodes des maillages non-structurés en aérodynamique numérique.

Nous sommes convaincus que le cycle de conférences pour lequel ces cours ont été écrits aura beaucoup de succès, et nous tenons à remercier nos deux organismes, l'Institut von Kármán et le NASA Ames Research Center pour l'organisation du cours. Nous remercions également l'AGARD et le Panel de la dynamique des fluides, son ancien Président Jim McCroskey, son Administrateur Winston Goodrich et sa Secrétaire Anne-Marie Rivault pour leurs encouragements et leur soutien professionnel.

Herman Deconinck et Tim Barth
Directeurs de cours

Special Course Staff

DIRECTORS/LECTURERS

Dr T. Barth
Mail Stop 202A-1
NASA Ames Research Center
Moffett Field, CA 94035-1000
United States

Dr H. Deconinck
von Kármán Institute for Fluid Dynamics
Chaussée de Waterloo, 72
1640 Rhode-Saint-Genèse
Belgium

LECTURERS

Mr. J.-D. Müller*
CFD Group
von Karman Institute for Fluid Dynamics
Chaussee de Waterloo, 72
1640 Rhode-Saint-Genèse
Belgium

Prof. J. Peraire**
Department of Aeronautics
Imperial College
Prince Consort Road
London SW7 2BY
United Kingdom

Mr. P. Vankeirsbilck*
CFD Group
Von Karman Institute for Fluid Dynamics
Chaussee de Waterloo, 72
1640 Rhode-Saint-Genèse
Belgium

Prof. T. Hughes
Division of Applied Mechanics
Durand Building
Room No. 281
Stanford University
Stranford, CA 94305-4040
United States

Prof. C. Johnson
Department of Mathematics
Chalmers University of Technology
University of Goeteborg
41296 Goeteborg
Sweden

Prof. R. Löhner
CMEE, SEAS
George Washington University
Washington D.C. 20052
United States

Prof. K. Morgan*
Department of Civil Engineering
University College of Swansea
University of Wales
Engineering Building
Singleton Park
Swansea SA2 8PP
United Kingdom

Prof. T. Tezduyar
University of Minnesota
Minnesota Supercomputer Institute
1200 Washington Avenue South
Minneapolis Minnesota 55415
United States

LOCAL COORDINATORS

Dr T. Barth
Mail Stop 202A-1
NASA Ames Research Center
Moffett Field, CA 94035-1000
United States

Dr H. Deconinck
von Kármán Institute for Fluid Dynamics
Chaussée de Waterloo, 72
1640 Rhode-Saint-Genèse
Belgium

PANEL EXECUTIVE

Dr W Goodrich

Mail from Europe:
AGARD—OTAN
Attn: FDP Executive
7, rue Ancelle
92200 Neuilly-sur-Seine
France

Mail from US and Canada:
AGARD—NATO
Attn: FDP Executive
Unit 21551
APO AE 09777

Tel: 33 (1) 47 38 57 75
Telex: 610176 (France)
Telefax: 33 (1) 47 38 57 99

* Presented lectures at the VKI only
** Presented lectures at NASA Ames only

Contents

FINITE ELEMENT METHODS FOR FLOW PROBLEMS

by

Claes Johnson
Dept. of Mathematics
Chalmers University of Technology
412 96 Göteborg
Sweden

0. INTRODUCTION

0.1. The SD-method

The purpose of this note is to give an overview of the Streamline Diffusion method (SD-method below), also referred to as Galerkin/Least Squares or SUPG (Streamline Upwind Petrov-Galerkin), as a general finite element method for hyperbolic type partial differential equations modelling convection-diffusion, compressible/ incompressible fluid flow or wave propagation. The SD-method, developed by Tom Hughes and the author together with co-workers during the eighties, gives the first general solution to the fundamental problem of constructing finite element methods for hyperbolic problems with the desired combination of good stability and high accuracy. The SD-method is a modified Galerkin method based on piecewise polynomial approximation with the following two basic modifications:

(0.1) a "streamline diffusion" modification of the test functions giving a weighted least squares control of the residual $R(U)$ of the finite element solution U.

(0.2) introduction of an artificial viscosity $\hat{\epsilon}$ of typically the form
$\hat{\epsilon} = \max(\epsilon, Ch|R(U)|/|\nabla U|, Ch^{3/2})$ or
$\hat{\epsilon} = \max(\epsilon, Ch^2|R(U)|/|U|, Ch^{3/2})$,
where h is the local mesh size and ϵ is the given viscosity and C denotes positive constants.

Further, the SD-method is characterized by

(0.3) consistent use of space-time finite element discretization for time dependent problems, with the basis functions being discontinuous in time and continuous (or discontinuous) in space.

Each of the modifications (0.1) and (0.2) increases the stability of the underlying Galerkin method in different ways and through different mechanisms. The first modification (0.1) gives control of the residual $R(U)$ of the finite element solution U, obtained by inserting the finite element solution into the exact differential equation, in a weighted L_2-norm with weight proportional to $h^{\frac{1}{2}}$. The second modification (0.2) gives L_2-control of all first derivatives of U with the weight $\hat{\epsilon}^{\frac{1}{2}}$. The two modifications (0.1) and (0.2) together add sufficient stability to guarantee e.g. the following important properties of the SD-method (when applicable): maximum norm stability, entropy consistency, error localization and monotone shock resolution. Further, the modifications play a crucial role in the adaptive SD-methods based on a posteriori error estimates developed recently. We recall that the finite element in its basic form, i.e. the standard Galerkin method with piecewise polynomial approximation, which has been so remarkably successful for elliptic and parabolic problems, does not work in general for hyperbolic problems: Unless the exact solution is globally smooth (or the mesh sufficiently refined everywhere) the standard Galerkin finite element solution will contain large spurious oscillations making the error large over large portions of space and time. The spurious oscillations reflect the lack of stability in the standard Galerkin method concerning the residual and derivatives of the finite element solution.

The space-time discretization (0.3) of the SD-method offers a great flexibility in the discretization, in particular through the possibility of using space-time meshes oriented in space-time. This gives a happy marriage of Eulerian and Lagrangean descriptions in the SD-method combining the advantages of each of the two approaches while avoiding the disadvantages of a full Euclidean or Lagrangean approach. Using space-time meshes oriented according to characteristics (or particle paths), we obtain a special variant of the SD-method, which we refer to as the Characteristic SD-method, or CSD-method for short, and which is of particular interest for incompressible flow including also free (or moving) boundaries. Orienting the mesh in space-time according to the nature of the solution, we obtain variants of the SD-method with features of shock-fitting methods.

The SD-method with basically the first modification (0.1) was introduced in the beginning of the eighties by Hughes and Brooks [HB1] for stationary convection diffusion problems. The theoretical analysis of the method in this form including also extensions to time-dependent problems based on space-time finite elements, was carried out in the early eighties by Johnson, Nävert and Pitkäranta [JN], [N], [JNP]. A main result of this analysis was to exhibit the role of the modification (0.1) to increase the stability of the Galerkin method. To emphasize the stability aspect, the term "streamline diffusion" directly related to stability was used (instead of streamline upwind [HB2]), with motivation from scalar convection problems where the least squares modification giving weighted L_2-control of the convective derivative of the finite element solution corresponds to introducing a diffusion term acting in the direction of the streamlines.

Extensions of the SD-method to the incompressible Navier-Stokes equations followed quickly ([HB2], [JS]) with further developments in recent years to a simple equal order velocity-pressure formulation, which has found wide applications including free boundary flow ([HS], [H3]).

A decisive step in the development of the SD-method was taken in the mid eighties with the introduction of the second modification (0.2) opening in particular the possibility of applying the SD-method to compressible flow including shocks ([H1-4], [JSz1], [JSzH]. It appears that the residual based artificial viscosity of the form $\hat{\epsilon} = Ch|R(U)|/|\nabla U|$, which has a new construction as compared to artificial viscosities used in finite difference/volume methods, is close to the minimal viscosity required to make a Galerkin method (including also (0.1)) work well for e.g. problems with shocks.

As indicated, space-time finite elements were introduced early in the SD-method, ([JNP], [N]), but the full exploitation of the generality of the space-time mesh was initiated later ([J3], [J4], [J6], [Ha2-3]) in the form of the CSD-method with applications to free boundary flow ([Ha3]). The idea of using space-time finite elements was taken up in ([Sh], [ShHJ]) and recently also in ([TLB]) with applications to free boundary flow. Another recent development concerns adaptive SD and CSD-methods, (see [EJ7-8], [J2], [JSz3], [HJ], [JH]), where in a new way the basic modifications (0.1) and (0.2) come to use e.g. to prove a posteriori error estimates underlying the adaptive procedures. As far as we know these results are the first to show that efficient and reliable error control is possible for hyperbolic problems.

The SD-method has now successfully been applied to, in particular, stationary and time dependent

(i) convection-diffusion problems,
(ii) incompressible Euler and Navier-Stokes equations,
(iii) compressible Euler and Navier-Stokes equations,
(iv) reactive compressible flow,
(v) second order wave equations ([J5], [Hu]).

A large number of and numerical theoretical results are availabe. The theoretical results in general go beyond previous results for other methods (including finite difference/finite volume methods) in generality and/or precision. Some basic features of the SD-method supported by theoretical and numerical evidence are as follows:

(0.4) High accuracy: $\mathcal{O}(h^{p+\frac{1}{2}})$ for smooth solutions with polynomials of degree p.

(0.5) Good stability: sharp monotone resolution of shocks and contact discontinuities.

(0.6) Localization: quick decay in upwind or crosswind propagation in scalar convection problems.

(0.7) Adaptive forms available based on sharp a posteriori error estimates.

(0.8) Limits of SD-solutions satisfy all entropy conditions (entropy consistency).

(0.9) Convergence for scalar conservation laws in several dimensions with piecewise polynomials of arbitrary degree on general meshes.

(0.10) Problems with free or movingboundaries may be approached in anatural way using the flexibility of the space-time mesh.

0.2. Comparison with other methods in CFD.

We now briefly compare the SD-method with other numerical methods in Computational Fluid Dynamics CFD such as (a) finite difference methods, (b) finite volume methods, (c) particle methods and (d) shock-fitting methods.

In each case (a)-(d) there is a closely related variant of the SD-method realizing the essential feature of the method (a)-(d) in a full finite element context.

(a) Finite difference methods

The classical finite difference method for fluid flow is obtained by modifying a centered difference approximation of convective terms by adding a combination of artificial viscosity of first order $-Ch\Delta_h U$ and third order $Ch^3\Delta_h^2 U$, where Δ_h and Δ_h^2 are discrete analogs of the Laplacian and bilaplacian, respectively, with a switch from first order to third order in regions of smoothness. Now, the standard Galerkin finite element method typically produces centered difference approximations to convective terms and the modificat[illegible]s (0.1) and (0.2) add artificial visc[illegible]ty in non-standard forms including in particular an automatic "switch" related to the size of the coefficient $\hat{\epsilon} = Ch|R(U)|/|\nabla U|$, which is typically of order $\mathcal{O}(h)$ close to shocks for example and of order $\mathcal{O}(h^2)$ or smaller in regions of smoothness. Thus, the SD-method contains basic features of centered difference schemes with artificial viscosity, but the technicalities are different in the SD-method. In particular, the SD-method avoids the use of fourth order dissipation and realizes the "switch" in viscosity in a full finite element context which is easy to implement and accessible for analysis.

(b) Finite volume methods

The SD-method may for non-viscous flows be used with discontinuous approximation in space. We refer to this variant of the SD-method, including the modifications (0.1) and (0.2) properly interpreted, as the Discontinuous Galerkin or DG-method. With piecewise constant approximation this leads to more or less classical finite volume methods depending on the context. With higher order discontinuous polynomial approximation the DG-method may be viewed as a higher order finite volume method. Thus the DG-method gives the natural formulation of higher order finite volume methods, which have not found a really satisfactory formulation within the classical context of finite volume methods involving ad hoc flux limiting, flux correction, post processing et.cet. For an analysis of the DG-method for conservation laws, see [JJ].

(c) Particle methods

With space-time meshes oriented according to particle paths the SD-method may be viewed as an approximate particle method with restart, or a variant of a method of the form "exact transport + projection".

However, the SD-method does not require exact transport or precise particle tracing. The mechanism is just the orientation of the space-time mesh in space-time which brings in a "particle feature" to the method, but precise orientation or "exact transport" is not required. The method will work also on non-oriented standard meshes, but the precision is improved by suitable approximate mesh orientation. Further, the SD-method offers a built in modified L_2-projection entering when the space mesh is changed at discrete time levels. This makes it possible to change the space mesh often without decreasing the accuracy by introducing oscillations or too much dissipation. Thus, the SD-method with space-time meshes oriented according to particle paths gives a general formulation of a "particle method" with the following advantages: precise particle tracing is not necessary and frequent restarts with remeshing in space are possible without essentially decreasing the accurary.

(d) Shock-fitting methods

Finally, orienting the space-time mesh according to solution features such as shocks rather than particle paths, which can naturally be realized in adaptive forms of the SD-method, we make contact with so called shock-fitting methods. In the SD-method the mechanism is again only the mesh orientation which may typically be obtained from derivative information from the computed solution.

To sum up, the SD-method characterized by (0.1)-(0.3) offers a large degree of generality and flexibility and may be viewed as giving very natural generalizations of all the classical techniques of CFD including finite difference, finite volume, particle and shock fitting methods. Thus, the SD-method brings a surprising degree of unity to CFD combining in particular the two worlds of Euclidean and Lagrangean methods, everything realized by a modified Galerkin approach with piecewise polynomials on space-time meshes.

0.3. The stability concept. Artificial viscosity.

A main result of the theoretical analysis of the SD-method is to exhibit the importance of a correct notion of stability both for stationary and time dependent problems. The classical stability concept for hyperbolic problems typically requiring in a time dependent problem an L_2-bound of the solution for positive time in terms of the L_2-norm of initial data at $t = 0$, or in the case of a stationary problem an L_2-bound of the solution in terms of the L_2-norm of the right-hand side, turns out not to be fully adequate. Instead, a new stability concept for the discrete problem involving in addition weighted L_2-control of the residual (through the streamline diffusion modification (0.1)) and first derivatives of the discrete solutions (through the artificial viscosity (0.2)) in terms of e.g. the L_2-norm of the initial data, turns out to be more appropriate and useful. In addition a new stability concept for linearized versions of the continuous problem is used in the context of deriving a posteriori error estimates underlying the adaptive versions or the SD-method. Thus, we emphasize the extreme importance of using relevant stability concepts for both the discrete and continuous problems and the new aspects of this fundamental problem brought to the light through the analysis of the SD-method.

0.4. Adaptive SD-methods. A posteriori and a priori error estimates.

The improved stability properties of the SD-method in particular make it possible to prove sharp a posteriori error estimates which may be used to design reliable and efficient adaptive variants of the SD-method. This seems to open for the first time the field of CFD to adaptive quantitative error control on mathematical basis with accurate resolution of fine scale features in e.g. boundary layers and shocks.

The a posteriori error estimates for the SD-method for hyperbolic flow problems typically have the form:

(0.10) $$\|e\|_{L_2} \leq C\|h^2\hat{\epsilon}^{-1}R(U)\|_{L_2}$$
$$\leq C\|\min(h|\nabla U|, h^{\frac{1}{2}}R)\|_{L_2},$$

where e is the error, $R(U)$ is the residual of the computed solution U, h is the mesh size and we used the definition of $\hat{\epsilon}$. This estimate should be compared to the typical corresponding estimate for the Standard Galerkin method for hyperbolic problems:

(0.11) $$\|e\|_{L_2} \leq C\|R\|_{L_2},$$

and the standard Galerkin method for elliptic problems:

(0.12) $$\|e\|_{L_2} \leq C\|h^2R\|_{L_2}.$$

The estimate (0.11) is in general useless for adaptive purposes since the right hand side of (0.11) will increase with decreasing mesh size until all features have been resolved. The estimate (0.12) for elliptic problems is sharp and may be used as the basis for reliable and efficient adaptive algorithms. Clearly (0.10) is a mixture of (0.11) and (0.12) using in particular the ellipticity in the SD-method introduced through the artificial viscosity $\hat{\epsilon}$, together with the particular design of $\hat{\epsilon}$.

The proof of the a posteriori error estimate (0.10) has the following structure:

1. Representation of the error in terms of the residual of the finite element solution and the solution of a continuous (linearized) dual problem.
2. Use of the Galerkin orthogonality built in the finite element method.
3. Interpolation estimates for the dual solution.
4. Strong stability estimates for the continuous dual problem.

We note the crucial roles played here by the residual, the Galerkin orthogonality and the strong stability of the continuous dual problem.

The typical a priori error estimate for the SD-method with piecewise polynomials of degree p takes the form

(0.13) $$\|e\|_{L_2} \leq C\|h^{p+\frac{1}{2}}D^{p+1}u\|_{L_2},$$

where u is the exact solution and $D^{p+1}u$ is the maximal modulus of derivatives of u of order $p+1$.

The proof of the a priori error estimate typically has the following structure:

1. Representation of the error in terms of the exact solution and a discretized dual problem.
2. Use of the Galerkin orthogonality to introduce the truncation error in the error representation.
3. Interpolation estimates for the truncation error.
4. Strong stability of the discrete dual problem.

We note the similarity in the structure of the proofs of the a priori and a posteriori error estimates, and also the differences: In the a priori case the key roles are played by the truncation error and the strong stability of the discrete problem, and in the a posteriori case these roles are taken by the residual and the stability of the continuous problem. Both the a priori and a posteriori error estimates are fundamental: The a priori error estimate shows that the discretization error (and the residual) will tend to zero with decreasing mesh size, and the a posteriori error estimate is the basis for adaptive quantitative error control.

0.5. Summary of the design principles of the SD-method

We recall some of the fundamental problems in CFD:

(0.14) design of artificial viscosity,
(0.15) unstructured meshes,
(0.16) how to combine Eulerian and Lagrangean methods.

Each of these problems has received massive attention over the years, but conclusive answers are still lacking to a large extent within the classical methodologies of CFD (finite difference methods, finite volume methods, particle and shock-fitting methods). Our main point is now that the SD-method with the three cornerstones (0.1)-(0.3) offers a new approach to each of the fundamental problems (0.14)-(0.16).

The construction of the artificial viscosity $\hat{\epsilon}$ in the SD-method with the dependence on the residual has a solid mathematical basis and appears to be close to the minimal artificial viscosity required to guarantee features such as entropy consistency, maximum norm stability, (almost) monotone shock resolution et cet, of prime interest in classical CFD. In addition, the artificial viscosity $\hat{\epsilon}$ of the SD-method makes it possible to design efficient adaptive SD-methods based on a posteriori error estimates. Thus, the SD-method offers a solution of the fundamental problem (0.14).

The space-time discretization of the SD-method has a maximal flexibility and appears to solve (0.16) as well as of course (0.15).

To sum up, it app[illegible]s that the finite element method (Galerkin + piecewise polynomial approximation) with the design principles (0.1)-(0.3), that is the SD-method, gives a fresh approach to the fundamental problems of CFD w[illegible] a surprising degree of unificati[illegible] opens many possibilities for the fut[illegible]

0.6. Outline

An outline of the cont[illegible] of this [illegible]te is as follows:

1. SD-methods f[illegible] [illegible] scalar linear convection [illegible]
2. SD-methods for time dependent scalar linear convection-diffusion.
3. The CSD-method for scalar linear convection-diffusion.
4. The SD and CSD-methods for the incompressible Euler and Navier-Stokes equations.
5. The SD-method for the compressible Euler equations.
6. The DG-method for conservation laws.
7. Adaptive SD-methods for Burgers' e[illegible]uation.
8. Numerical [illegible]esults
9. Conclusion. Prospects for the fut[illegible]

1. SD-METHODS FOR STATIONARY LINEAR SCALAR CONVECTION-DIFFUSION

1.1. Introduction

As a basic model problem we shall consider the following stationary linear scalar convection-diffusion problem: Find the concentration $u = u(x)$ such that

$$(1.1a)\qquad \beta\cdot\nabla u + \alpha u - \operatorname{div}(\epsilon\nabla u) = f \quad \text{in } \Omega,$$

$$(1.1b)\qquad u = g \quad \text{on } \Gamma,$$

where Ω is a bounded convex domain in the plane $\mathbb{R}^2$, $\beta = \beta(x) = (\beta_1(x), \beta_2(x))$ is a given velocity field, $\alpha = \alpha(x)$ is an absorption coefficient, $\epsilon = \epsilon(x)$ is a positive viscosity coefficient, which we typically assume to be "small" in a sense to be made precise, and $f \in L_2(\Omega)$ and $g \in L_2(\Gamma)$ is a given production term and boundary data. Note that (1.1) is the basic model for processes involving convection-absorption-diffusion-reaction with a very large range of applicability. All coefficients β, α and ϵ are assumed to be "sufficiently smooth" according to requirements made more precise below. We shall assume that

$$(1.2)\qquad (-\tfrac{1}{2}\operatorname{div}\beta + \alpha)(x) \geq \alpha_0 > 0,$$

where α_0 is a positive constant. This assumption is not essential in many cases where it is sufficient to assume $(-\frac{1}{2}\operatorname{div}\beta + \alpha)$ to be bounded from below (cf. [JNP]). If $\epsilon = 0$ then (1.1) reduces to

$$(1.3a)\qquad \beta\cdot\nabla u + \alpha u = f \quad \text{in } \Omega,$$

$$(1.3b)\qquad u = g \quad \text{on } \Gamma_-,$$

where now the boundary condition is only enforced on th inflow part $\Gamma_- = \{x \in \Gamma: n(x)\cdot\beta(x) < 0\}$ of Γ, where $n(x)$ is the outward unit normal to Γ. The solution u of (1.1) will typically have an outflow layer of width $\mathcal{O}(\epsilon)$ at the outflow part $\Gamma_+ = \{x \in \Gamma: n(x)\cdot\beta(x) > 0\}$ of the boundary and may have internal layers of wi[illegible]th $\mathcal{O}(\sqrt{\epsilon})$ along streamlines $x(t)$ given by β, i.e. solutions of

(1.4a) $\frac{dx}{dt} = \beta(x) \qquad t > 0,$

(1.4b) $x(0) = \bar{x},$

e.g. if the inflow data g is discontinuous or if f is discontinuous across a streamline. Thus, the solution u of (1.1) typically has features on the different scales $\mathcal{O}(1)$, $\mathcal{O}(\sqrt{\epsilon})$ and $\mathcal{O}(\epsilon)$. The essential difficulty in the numerical solution of (1.1) is now the presence of small scale features in u, which if not resolved may lead to spurious oscillations in a standard Galerkin approach. Let us remark here, anticipating the discussion of adaptive SD-methods below, that even if an adaptive process may lead to a final mesh where all features of the exact solution are resolved so that on the final mesh standard Galerkin would be possible to use, the meshes in the initial stages of the adaptive process would not be fine enough to resolve all features, in which case we need robust discretization methods like the SD-method which is able to produce good results without requiring global resolution. The only way to avoid this would be to use an intitial mesh refined uniformly everywhere which is not cost effective, or just impossible in most interesting cases.

The basic problem is now to design and analyze a finite element method for (1.1) which is higher than first order accurate, and which has good stability properties so that resolution of all features will not be necessary to produce reasonable results; an unresolved local feature should not degrade the error globally but only locally. We shall see that the SD-method gives a very satisfactory solution to this problem. Note that if we give up the requirement of accuracy higher than first order, the solution is immedeate: Just take $\hat{\epsilon} = C|\beta|h$ as artificial viscosity and apply the standard Galerkin method. However, this method in general adds much too much articial viscosity and will require excessive mesh refinement to give reasonable accuracy, and thus cannot be considered to give a satisfactory solution in general. Note that this simple method corresponds to the classical artificial viscosity method or upwind method in finite difference theory.

In the formulation of the SD-method below the given viscosity ϵ will, if the mesh size is not small enough, be replaced by the artificial viscosity $\hat{\epsilon}$ depending on the computed solution U and the mesh size h. It is convenient (and natural) to split the total error $e = u - U$ into

(1.5) $\quad u - U = (u - \hat{u}) + (\hat{u} - U),$

where $\hat{u}$ satisfies (1.1) with ϵ replaced by $\hat{\epsilon}$, i.e., $\hat{u}$ is the solution of the continuous problem

(1.6a) $\quad \beta\cdot\nabla\hat{u} + \alpha\hat{u} - \operatorname{div}(\hat{\epsilon}\nabla\hat{u}) = f \quad \text{in } \Omega,$

(1.6b) $\quad \hat{u} = g \quad \text{on } \Gamma.$

Now, $u - \hat{u}$ is the difference between the solutions of two continuous problems with different viscosity ϵ and $\hat{\epsilon}$, and $\hat{u} - U$ is the discretization error related to (1.6) with now $\hat{\epsilon}$ considered to be given. The advantages of the splitting (1.5) are as follows: The estimation of the discretization error $\hat{u} - U$ will concern a linear problem, whereas the full problem is nonlinear since $\hat{\epsilon}$ depends on U. Further in an adaptive approach the perturbation error $u - \hat{u}$ may be controled by controlling the difference $\hat{\epsilon} - \epsilon$, cf below.

1.2. Finite element prerequisites

For simplicity we shall below restrain the detailed presentation to the simplest possible finite elements: piecewise linear functions on triangles (two dimensions) or tetrahedrons (three dimensions). This is not essential and all methods to be presented have natural generalizations to general piecewise polynomial approximation.

Below Ω will denote a bounded domain in $\mathbb{R}^d$, $d = 2,3$, with polygonal or polyhedral bundary Γ. By $T_h = \{K\}$ we will denote a triangulation of Ω, i.e., a subdivision of Ω into triangles $(d = 2)$ or tetrahedrons $(d = 3)$ K such that different elements K which intersect, share either a vertex or a side $(d = 2)$ or a vertex, edge or face $(d = 3)$. The local mesh size of T_h will be given by the mesh function $h(x)$ satisfying

(1.7) $c_1 h_K \le h(x) \le c_2 h_K$ for $x \in K$,

where c_1 and c_2 are positive constants independent of T_h and h_K is the diameter of K. We shall further assume that the smallest angle of the elements $K \in T_h$ are uniformly bounded below.

The basic finite element space used below will now be the following: $V_h = \{v\colon v$ is continuous on $\bar\Omega \equiv \Omega \cup \Gamma$ and v is linear on K, $\forall K \in T_h\}$, i.e. the space of continuous piecewise linear functions on Ω. By π_h we shall denote a standard interpolation operator into V_h defined by nodal interpolation (or variants thereof involving local smoothing) or through L_2-projection. We shall assume that π_h satisfies the following interpolation estimates: There are constants C independent of T_h such that

(1.8a) $$\|D^k(v - \pi_h v)\|_{L_2(\Omega)} \le C\|h^{m-k}D^m v\|_{L_2(\Omega)} \quad 0 \le k \le m \le 2,$$

(1.8b) $$\|v - \pi_v v\|_{L_2(\partial K)} \le Ch^{m-\frac{1}{2}}\|D^m v\|_{L_2(\tilde K)},$$
$$1 \le m \le 2,\ K \in T_h,$$

where $\tilde K$ is the union of K and the nearest neighbors of K if $m = 1$, and $\tilde K = K$ if $m = 2$, and

$$D^m v = \max_{|\alpha|=m} D^\alpha v,$$

with $D^\alpha v = \dfrac{\partial^{\alpha_1+\ldots+\alpha_d} v}{\partial x_1^{\alpha_1}\ldots\partial x_d^{\alpha_d}}$, $|\alpha| = \alpha_1 + \ldots + \alpha_d$, using standard multiindex notation.

We further recall the following inverse estimate: There is a constant C such that for all $v \in V_h$

(1.9c) $$\|\nabla v\|_{L_2(K)} \le Ch_K^{-1}\|v\|_{L_2(K)}, \quad K \in T_h.$$

We shall below use the following notation

$$(v,w) = \int_\Omega vw\,dx,$$

$$(\nabla v, \nabla w) = \int_\Omega \nabla v\cdot\nabla w\,dx,$$

$$\|v\| = (v,v)^{\frac{1}{2}},\quad \|\nabla v\| = (\nabla v, \nabla v)^{\frac{1}{2}}.$$

1.3. Formulation of the SD-method for (1.1) and (1.3).

To define the SD-method for (1.1) with $g = 0$ for simplicity, let $\mathring{V}_h \subset H_0^1(\Omega)$ be the standard finite element space of piecewise linear functions on a triangulation $T_h = \{K\}$ of Ω which vanish on Γ. The SD-method for (1.1) now reads as follows: Find $U \in \mathring{V}_h$ such that

(1.10) $$a(U,v) = L(v) \quad \forall v \in \mathring{V}_h,$$

where

$$a(w,v) =$$
$$= (\beta\cdot\nabla w + \alpha w, v + \delta(\beta\cdot\nabla v + \alpha v))$$
$$+ (\epsilon\nabla v, \nabla v),$$
$$L(v) = (f, v + \delta(\beta\cdot\nabla v + \alpha v)),$$

where

(1.11a) $$\delta = C_1 \max(h - \frac{\epsilon}{|\beta|}, 0)/|\beta|,$$

(1.11b) $\hat{\epsilon} = \hat{\epsilon}(h, R(U)) =$

$$\max(\epsilon, C_2 h \frac{R(U)}{|\nabla U|+h}, C_3 h^{3/2}),$$

$$R(U) = R_1(U) + R_2(U),$$

(1.11c) $R_1(U) =$

$$|\beta\cdot\nabla U + \alpha U - \operatorname{div}(\hat{\epsilon}\nabla U) - f| \text{ in } K,$$

(1.11d) $R_2(U)|_K =$

$$\max_{S\subset\partial K} \max_S \frac{1}{2} |[\hat{\epsilon}\, n_S\cdot\nabla U]|/h_K,$$

where $[\hat{\epsilon}\, n_S\cdot\nabla U]$ denotes the jump in the quantity $\hat{\epsilon}\, n_S\cdot\nabla U$ across a triangle side S (in the interior of Ω) with normal n_S.

Further, we use the convention that integrals over Ω containing second derivatives are to be interpreted as a sum of integrals over the elements K, and we also recall that $h(x) \sim h_K$ for $x \in K$. The constants C_1 and C_2 only depend on the type of polynomial approximation and the shape of the elements. For $p = 1$ and triangular elements one may take $C_1 = C_2 \sim 0.5$. The constant C_3 may normally be chosen to be zero, but there is (at least in theory) a reason to have $C_3 > 0$. Note that with ϵ very small, we have $\delta \sim \frac{h}{2|\beta|}$ and $\hat{\epsilon} \sim \frac{hR(U)}{2|\nabla U|}$ (if C_3 is small), while $\delta = 0$ for $\hat{\epsilon} \geq h|\beta|$ and $\hat{\epsilon} = \epsilon$ if $\epsilon \geq \max(C_2 hR(U)/(|\nabla U|+h), C_3 h^{3/2})$. Notice further that in principle $\hat{\epsilon}$ is implicitely defined through (1.11b) since R(U) depends on $\hat{\epsilon}$ through (1.11c). This may be simplified in practice by choosing $\hat{\epsilon} = 0$ in (1.11c,d).

Notice that (1.10) is obtained multiplying (1.1a) by $\delta(\beta\cdot\nabla v+\alpha v)$, where the $\hat{\epsilon}\delta$-term may be omitted in the case of piecewise linear approximation in space, see [JNP].

Let us also formulate the SD-method for the reduced problem (1.3) as follows using the space V_h of continuous piecewise linear functions on Ω without any boundary conditions enforced and imposing the inflow condition (1.3b) weakly: Find $U \in V_h$ such that

(1.12) $a_0(U,v) = L_0(v) \quad \forall v \in V_h,$

where

$$a_0(w,v) = a(w,v) - \int_{\Gamma_-} wv\beta\cdot n ds,$$

$$L_0(v) = L(v) - \int_{\Gamma_-} gv\beta\cdot n ds.$$

where δ and $\hat{\epsilon}$ are defined by (1.11) with $\epsilon = 0$.

Let us note that with $C_1 = C_2 = C_3 = 0$, in (1.10) and (1.12), we get the standard Galerkin method (with weakly imposed boundary conditions in the case (1.12)), while the choice $C_1 = 0$ and setting $\hat{\epsilon} = h|\beta|$ would give the classical artificial viscosity method corresponding to a full upwind approximation of the convective derivative (with piecewise linears). In the following table we roughly summarize the characteristics of the three methods discussed with a plus or minus sign indicating a satisfactory quality or not.

	Stability	Accuracy
Standard Galerkin $C_1=C_2=C_3=0$	-	+
Classical artificial viscosity $C_1=0$, $\hat{\epsilon}=h\|\beta\|$,	+	-
Streamline Diffusion $C_1\sim C_2\sim 0.5$, C_3 small	+	+

1.5. The basic stability estimates for the SD-method

The motivation for the two modifications (0.1) and (0.2) characterizing the SD-method is to increase stability (without sacrificing accuracy). Let us now derive the basic stability estimates for the SD-method (1.10) and (1.12). These estimates will follow from the following coercivity properties of the bilinear forms $a(v,v)$ and $a_0(v,v)$: There are positive constants c depending on α and α_0 such that

$$(1.13) \qquad a(v,v) \geq c|||v|||^2 \quad \forall v \in \hat{V}_h,$$

$$(1.14) \qquad a_0(v,v) \leq c|||v|||_0^2 \quad \forall v \in V_h,$$

where.

$$|||v||| = (\|v\|^2 + \|\hat{\epsilon}^{\frac{1}{2}}\nabla v\|^2 + \|\delta^{\frac{1}{2}}\beta\cdot\nabla v\|^2)^{\frac{1}{2}},$$

$$|||v|||_0 = (|||v|||^2 + \frac{1}{2}\int_\Gamma v^2|\beta\cdot n|\,ds)^{\frac{1}{2}}.$$

To prove (1.13) we note that by Green's formula

$$\int_\Omega \beta\cdot\nabla v\, v dx = -\int_\Omega \beta\cdot\nabla v v\, dx$$

$$-\int_\Omega \operatorname{div}\beta\, v^2 dx + \int_\Gamma \beta\cdot n\, v^2 ds,$$

which gives

$$(\beta\cdot\nabla v + \alpha v, v)$$

$$= ((-\frac{1}{2}\operatorname{div}\beta + \alpha)v, v) + \frac{1}{2}\int_\Gamma v^2\beta\cdot n ds,$$

from which (1.13) and (1.14) directly follow recalling (1.2). We note the improved stability of the SD-method as compared to the standard Galerkin method satisfying (1.13) and (1.14) with $\delta = 0$ and $\hat{\epsilon} = \epsilon$.

1.6. An a priori error estimate for the SD-method

Let us prove an a posteriori error estimate for the SD-method (1.12). Without loosing the essential part of the argument we shall then assume for simplicity that

$\hat{\epsilon} = 0$ and $\alpha = 0$. Recalling (1.3) we then have

$$a_0(u,v) = L_0(v) \qquad \forall v \in V_h,$$

which together with (1.12) gives the error equation

$$(1.15) \qquad a_0(u - U, v) = 0 \qquad \forall v \in V_h.$$

Recalling the stability estimate (1.14) we get using (1.15) with $v = \pi_h u - U$, where $\pi_h u \in V_h$ is the interpolant of u,

$$c|||u - U|||_0^2 \leq a_0(u - U, u - U)$$

$$= a_0(u - U, u - \pi_h u)$$

$$+ a_0(u - U, \pi_h u - U)$$

$$= a_0(u - U, u - \pi_h u)$$

$$\leq \|\delta^{\frac{1}{2}}\beta\cdot\nabla(u - U)\|\cdot\|\delta^{-\frac{1}{2}}(u - \pi_h u)\|$$

$$+ \|\delta^{\frac{1}{2}}\beta\cdot\nabla(u-U)\|\|\delta^{\frac{1}{2}}\beta\cdot\nabla(u-\pi_h u)\|$$

$$+(\int_\Gamma(u-U)^2|\beta\cdot n|ds)^{1/2}$$

$$(\int_{\Gamma_-} (u-\pi_h u)^2|\beta\cdot n|ds)^{1/2}$$

$$\leq \frac{c}{2}|||u-U|||_0^2 + C\|\delta^{-\frac{1}{2}}(u - \pi_h u)\|^2$$

$$+ C\|\delta^{\frac{1}{2}}\beta\cdot\nabla(u-\pi_h u)\|^2$$

$$+ (\int_{\Gamma_-} (u-\pi_h u)^2|\beta\cdot n|ds)^{1/2}.$$

Recalling (1.8) this proves that

(1.16a) $$|||u-U|||_0 \le C\|h^{3/2}D^2u\|,$$

which is the basic a priori error estimate for the SD-method (1.12).

We note that (1.16a), in addition to the slightly non-optimal L_2-estimate

(1.16b) $$\|u-U\| \le C\|h^{3/2}D^2u\|,$$

contains the estimate $\|\delta^{\frac{1}{2}}\beta\cdot\nabla(u-U)\| \le C\|h^{3/2}D^2u\|$, that is assuming for simplicity that $h(x) = h$ constant and $|\beta| \sim 1$, we have the optimal estimate

(1.16c) $$\|\beta\cdot\nabla(u-U)\| \le C\|hD^2u\|$$

for the convection operator $\beta\cdot\nabla$ applied to error $u - U$. In particular we have for the residual $R(U) = f-\beta\cdot\nabla U-\alpha U = \beta\cdot\nabla(u-U) + \alpha(u-U)$ the following optimal estimate

(1.16d) $$\|R(U)\| \le C\|hD^2u\|.$$

The estimates (1.16a-d) for the SD-method should be compared with the following estimates for the standard Galerkin method

(1.16e) $$\|u-U\| \le C\|hD^2u\|,$$
(1.16f) $$\|R(U)\| \le C\|D^2u\|,$$

which are seriously non-optimal.

The error estimates (1.16a-d) are meaningful only if h is small enough to resolve all features of u requiring h to be smaller than ϵ in e.g. outflow layers. In case some features of u are not resolved we may replace for instance (1.16b) by a localized analog of essentially the form

(1.16g) $$\|u-U\|_{L_2(\Omega')} \le C\|h^{3/2}D^2u\|_{L_2(\Omega'')}$$

where $\Omega' \subset \Omega'' \subset \Omega$ and u is smooth in Ω''. The truncated domain Ω'' is obtained from Ω by downwind or crosswind "cut-off" with cut-off distances of order $\mathcal{O}(h)$ and $\mathcal{O}(h^{3/4})$, respectively. This means that unresolved layers do not degrade the error at the indicated distances in upwind or crosswind directions. This is contrast to the standard Galerkin where an unresolved layer may degrade the error in the whole domain. Proofs of the indicated cut-off results for the SD-method, which are based on the improved stability resulting from the streamline diffusion modification, are given in [JNP].

1.7. An a posteriori error estimate for the SD-method

Let us now prove a basic a posteriori error estimate for the SD-method (1.12). In this case the artificial viscosity $\hat{\epsilon}$ plays a crucial role, while we may take $\delta = 0$ and also $\alpha = 0$ without loosing the essentials of the argument. The discrete problem thus reads: Find $U \in V_h$ such that

(1.17) $$(\beta\cdot\nabla U,v) + (\hat{\epsilon}\nabla U,\nabla v) - \int_{\Gamma_-} Uv\beta\cdot n ds = (f,v) - \int_{\Gamma_-} gv\beta\cdot n ds \quad \forall v \in V_h.$$

The perturbed continuous problem with solution $\hat{u}$ takes the following variational form: Find $\hat{u} \in H^1(\Omega)$ such that

(1.18) $$(\beta\cdot\nabla\hat{u},v) + (\hat{\epsilon}\nabla\hat{u},\nabla v) - \int_{\Gamma_-} \hat{u}v\beta\cdot n ds = (f,v) - \int_{\Gamma_-} gv\beta\cdot n ds, \quad \forall v \in H^1(\Omega).$$

Let us now introduce the following continuous dual problem: Find $\varphi \in H^1(\Omega)$ such that

(1.19) $$-(v, \beta\cdot\nabla\varphi) - ((\text{div }\beta)v, \varphi) + (\hat{\epsilon}\nabla v, \nabla\varphi) + \int_{\Gamma_+} v\varphi\beta\cdot n\, ds = (v, \hat{e}) \quad \forall v \in H^1(\Omega),$$

$H^1(\Omega)$ where $\hat{e} = \hat{u} - U$ and $\Gamma_+ = \{x: \beta(x)\cdot n(x) > 0\}$. Taking $v = \hat{e}$ in (1.19) we get the following error representation formula

$$\|\hat e\|^2 = -(\hat e, \beta\cdot\nabla\varphi) - ((\operatorname{div}\beta)\hat e, \varphi)$$

$$+ (\hat\epsilon\nabla\hat e, \nabla\varphi) + \int_{\Gamma_+} \hat e\varphi\beta\cdot n ds$$

$$=(\beta\cdot\nabla\hat e, \varphi)+(\hat\epsilon\nabla\hat e, \nabla\varphi)-\int_{\Gamma_-} \hat e\varphi\beta\cdot n\, ds$$

$$= (\beta\cdot\nabla\hat u, \varphi) + (\hat\epsilon\nabla\hat u, \nabla\varphi)$$

$$- \int_{\Gamma_-} \hat u\varphi\beta\cdot n\, ds - (\beta\cdot\nabla U, \varphi)$$

$$- (\hat\epsilon\nabla U, \nabla\varphi) + \int_{\Gamma_-} U\varphi\beta\cdot n\, ds$$

$$= (f,\varphi) - \int_{\Gamma_-} g\varphi\beta\cdot n\, ds - (\beta\cdot\nabla U, \varphi)$$

$$- (\hat\epsilon\nabla U, \nabla\varphi) + \int_{\Gamma_-} U\varphi\beta\cdot n\, ds$$

$$= (f, \varphi-\Phi) + \int_{\Gamma_-} (U-g)(\varphi-\Phi)\beta\cdot n\, ds$$

$$- (\beta\cdot\nabla U, \varphi-\Phi) - (\hat\epsilon\nabla U, \nabla(\varphi-\Phi))$$

so that

(1.20) $$\|\hat e\| = (f - \beta\cdot\nabla U, \varphi - \Phi) + \int_{\Gamma_-} (U-g)(\varphi-\Phi)\beta\cdot n\, ds - (\hat\epsilon\nabla U, \nabla(\varphi-\Phi)),$$

where we integrated by parts and used (1.17), (1.18) and $\Phi \in V_h$ is arbitrary. Integrating by parts on each element K in the third term on the right hand side of (1.20), we get

(1.21) $$\|\hat e\|^2 =(f-\beta\cdot\nabla U+\operatorname{div}(\hat\epsilon\nabla U),\varphi-\Phi) - \sum_K \int_{\partial K} \hat\epsilon \frac{\partial U}{\partial n_K} (\varphi-\Phi)ds$$

$$+ \int_{\Gamma_-} (U-g)(\varphi-\Phi)\beta\cdot n\, ds$$

$$= (f-\beta\cdot\nabla U + \operatorname{div}(\hat\epsilon\nabla U), \varphi-\Phi)$$

$$+ \sum_S \int_S [\hat\epsilon \frac{\partial U}{\partial n_S}](\varphi-\Phi)ds$$

$$- \int_{\Gamma} \hat\epsilon \frac{\partial U}{\partial n} (\varphi-\Phi)ds$$

$$+ \int_{\Gamma_-} (U-g)(\varphi-\Phi)\beta\cdot nds$$

$$= (\bar R_1(U), \varphi-\Phi)$$

$$+ \sum_K \int_{\partial K} h_K \bar R_2(U)(\varphi-\Phi)ds,$$

where

$$\bar R_1(U)|_K = (f-\beta\cdot\nabla U + \operatorname{div}(\hat\epsilon\nabla U))|_K \quad K \in T_h,$$

$$h_K\bar R_2(U)=\begin{cases} \frac{1}{2} [\hat\epsilon\, n_S \cdot \nabla U] & \text{on } \partial K, \\ -\hat\epsilon \frac{\partial U}{\partial n} & \text{on } S \subset \partial K, \\ -\hat\epsilon \frac{\partial U}{\partial n} +(U-g)\beta\cdot n & \text{on } S\subset\partial K, \end{cases}$$

if $\partial K \cap \Gamma = \emptyset$, $S \subset \Gamma_+$, or $S \subset \Gamma_-$, respectively.

We shall now choose $\Phi = \pi_h\varphi$ and use the following strong stability estimate for φ to estimate $\varphi-\Phi$ via (1.8) (for a proof we refer to [EJ7]):

<u>Lemma 1.1.</u> Under sufficient regularity assumptions on β and $\hat\epsilon$ there is a constant C such that if φ satisfies (1.19), then

$$\|\varphi\| + \|\epsilon^{\frac{1}{2}}\nabla\varphi\| + \|\operatorname{div}(\hat\epsilon\nabla\varphi)\|$$

$$+ \|\beta\cdot\nabla\varphi + \varphi \operatorname{div}\beta\| \le C\|\hat e\|$$

and

$$\|\hat{\epsilon}D^2\varphi\| \leq C\|\hat{e}\|.$$

Recalling the error representation formula (1.21) we now get

$$(1.22) \qquad \|\hat{e}\|^2 \leq C\|h^2\hat{\epsilon}^{-1}R\|,$$

where $R = R_1 + R_2$, $R_1 = |\bar{R}_1|$ and

$$R_2|_K = \max_{S\subset\partial K} \max_{S} |\bar{R}_2|.$$

We thus obtain the following a posteriori error estimate for (1.12) recalling that $\hat{\epsilon} = \max(C_2hR(U)/|\nabla U|, C_3h^{3/2})$:

<u>Theorem 1.1.</u> There is a constant C such that if U and $\hat{u}$ satisfy (1.17) and (1.18), then $\hat{e} = \hat{u} - U$ satisfies

$$(1.23) \qquad \|\hat{e}\| \leq \|E(h,U,f)\|,$$

where

$$E(h,U,f) = Ch^2\hat{\epsilon}^{-1}R$$

$$= C\min(C_2^{-1}h|\nabla U|, C_3^{-1}h^{\frac{1}{2}}R).$$

We note that by (1.16d) the quantity $E(h,U,f)$ appears to be of order $\mathcal{O}(h^{3/2})$ in regions of smoothness of the exact solution and of order $\mathcal{O}(1)$ close to a discontinuity. Thus, (1.23) appears to be as sharp as possible. For numerical results for adaptive algorithms based on (1.23) we refer to [EJ7]. In such algorithms one seeks, in an iterative process, a mesh $T_h = \{K\}$ such that

$$E(h,U,f)|_K h_K \cong TOL/\sqrt{N_h}, \forall K \in T_h,$$

where $TOL > 0$ is a tolerance given by the user and N_h is the number of elements K in T_h, corresponding to equidistribution of the element contributions in the right hand side of (1.23).

With $\epsilon > 0$, we may add the condition $\hat{\epsilon} = \epsilon$ in the adaptive process, corresponding to resolution of all details, see Section 8.

2. SD-METHODS FOR TIME-DEPENDENT LINEAR SCALAR CONVECTION DIFFUSION.

2.1. A model problem

We shall now consider the SD-method for the following time dependent model problem:

$$(2.1a)\quad u_t + \beta\cdot\nabla u - \operatorname{div}(\epsilon\nabla u) = f \quad \text{in } \Omega\times I,$$

$$(2.1b)\quad u = 0 \quad \text{on } \Gamma\times I,$$

$$(2.1c)\quad u(\cdot,0) = u \quad \text{in } \Omega,$$

where $u_t = \frac{\partial u}{\partial t}$, $\beta = \beta(x,t)$ is a given smooth velocity field, $\epsilon(x,t)$ is a positive viscosity, Ω is a bounded polygonal domain in $\mathbb{R}^2$ with boundary Γ, $I = (0,T)$ with $T > 0$ is a given time interval, and $f = f(x,t)$ and $u_0 = u_0(x)$ are given data.

2.2. Space-time discretization

Let $0 = t_0 < t_1 < t_2 <\ldots< t_n <\ldots t_N = T$ be a sequence of discrete time levels, set $I_n = (t_n, t_{n+1})$, $k_n = t_{n+1} - t_n$ and introduce the space-time strips or "slabs" $S_n = \Omega\times I_n$. Let for $n = 1,\ldots, N$, $T_n = \{K\}$ be a finite element subdivision of S_n into space-time elements K of diameter h_K with corresponding mesh function $h_n(x,t)$ and let $V_n \subset \hat{H}^1(S_n) = \{(v \in H^1(S_n)$: $v = 0$ on $\Gamma\times I_n\}$ be a corresponding finite element space, and define

$$V = \prod_{n=1}^{N} V_n$$

$$\equiv \{v: v|_{S_n} \in V_n, \qquad n = 1,\ldots, N\}.$$

The space-time mesh $T_n = \{K\}$ on S_n may typically consist of standard tensor product elements $K = \kappa \times I_n$, where κ is a triangle or quadrilateral in $\mathbb{R}^2$, or "tilted" such elements (see below), or tetrahedrons K of height k_n. The mesh T_n may also be more general with time steps variable in space.

We notice that $v \in V$ may be discontinuous in time at the discrete time levels t_n and to account for this fact we define

$$v^n_{\pm}(\cdot) = \lim_{s \to 0^+} v(\cdot, t_n \pm s),$$

$$[v^n] = v^n_+ - v^n_-.$$

Using the proper generalization of (1.9) and (1.12) to the time-dependent case, we shall now seek an approximation $U \in V$.

2.3. The SD-method on general space-time meshes.

We now formulate the SD-method for (2.1) as follows: Find $U \in V$ such that

(2.2) $\qquad a(U,v) = L(v) \quad \forall v \in V,$

where

$$a(w,v) = \sum_{n=0}^{N-1} a_n(w,v)$$

$$+ \sum_{n=1}^{N-1} \int_\Omega [w^n] v^n_+ \, dx + \int_\Omega w^0_+ v^0_+ dx,$$

$$a_n(w,v) =$$

$$= (w_t + \beta\cdot\nabla w, v + \delta(v_t + \beta\cdot\nabla v))_n$$

$$+ (\hat{\epsilon}\nabla w, \nabla v)_n + (\hat{\epsilon} w_t, v_t)_n,$$

$$L(v) = \sum_{n=0}^{N-1} (f, v + \delta(v_t + \beta\cdot\nabla v))_n + (u_0, v^0_+)$$

$$(v,w)_n = \int_{I_n} (v,w)dt,$$

with

$$\delta = C_1(k_n^{-2} + h_n^{-2}|\beta|^2)^{-\frac{1}{2}} \quad \text{on } S_n,$$

$$\hat{\epsilon} = \max(\epsilon, C_2 h R(U)/|\nabla U|, C_3 h^{3/2}),$$

$$R(U) = \sum_{j=1}^{3} R_j(U),$$

$$R_1(U) = U_t + \beta\cdot\nabla U - \mathrm{div}(\hat{\epsilon}\nabla U) \quad \text{on } K,$$

$$R_2(U)|_K = \max_{\partial K} \frac{1}{2} |[\hat{\epsilon} n_x \cdot \nabla U]|/h_K,$$

$$R_3(U) = |[U^n]|/k_n \quad \text{on } S_n,$$

where $[\hat{\epsilon} n_x \cdot \nabla U]$ denotes the jump in the quantity $\hat{\epsilon} n_x \cdot \nabla U$ across a side S of K with normal $n = (n_x, n_t)$, and where $[U^n]$ is extended to S_n as a constant in t.

The basic a priori error estimate for (2.2) analogous to (1.16a) reads in the case $\hat{\epsilon} = \epsilon = 0$ with piecewise linear approximation in x and t,

(2.3) $\qquad |||e||| \le C\|h^{3/2} D^2 u\|_{L_2(Q)}$

where

(2.3) $\qquad |||v||| = (\|v\|^2_{L_2(Q)}$

$$+ \sum_{n=0}^{N-1} \|\delta^{\frac{1}{2}}(v_t + \beta\cdot\nabla v)\|^2_{L_2(S_n)}$$

$$+ \sum_{n=0}^{N-1} \|[v^n]\|^2_{L_2(\Omega)})^{1/2},$$

where $h(x,t) \sim \mathrm{diam}\, K$ if $(x,t) \in K$ and $D^2 u = D^2_{(x,t)} u$ is the maximal partial derivative of u of order 2 with respect to

(x,t). We note the presence in $|||v|||$ of the jumps $[v^n]$.

3. THE CSD-METHOD FOR LINEAR SCALAR CONVECTION-DIFFUSION PROBLEMS

3.1. A model problem

We now turn to a particular variant of the SD-method (2.2) for time-dependent linear scalar convection diffusion obtained by orienting the mesh approximately along characteristics locally in time. As indicated, we will refer to this method as the characteristic SD-method or CSD-method. This method combines the advantages of both Eulerian and Lagrangean techniques without suffering from the disadvantages of either approach, i.e. oscillations or excessive artificial diffusion in Eulerian methods and mesh distortion difficulties in Lagrangean methods.

To isolate the essential features, we shall as a model problem consider the following pure initial value problem:

$$(3.1a)\quad u_t+\beta\cdot\nabla u-\mathrm{div}(\epsilon\nabla u)=f(x,t) \quad \text{in } Q,$$

$$(3.1b)\qquad u(\cdot,0) = u_0 \qquad \text{in } \mathbb{R}^2,$$

where $Q = \mathbb{R}^2\times(0,T)$, and $\beta\colon Q\to\mathbb{R}^2$ is a smooth velocity field with

$$(3.2)\qquad \mathrm{div}\,\beta = 0 \quad \text{in } Q,$$

and f and u_0 are given functions with compact support in $\bar{Q} \equiv \mathbb{R}\times[0,\infty)$ and $\mathbb{R}^2$ respectively and $\epsilon \geq 0$ is constant. The restrictions to a pure initial value problem, a divergence-free velocity field β and constant ϵ are not essential.

We recall that the <u>characteristics</u> of the differential operator $(\frac{\partial}{\partial t}+\beta\cdot\nabla)$ in (3.1a) are curves $x = x(\chi,\tau)$ given by

$$(3.3a)\qquad \frac{dx}{d\tau} = \beta(x,\tau) \quad \tau > 0,$$

$$(3.3b)\qquad x(\chi,0) = \chi.$$

We note that

$$(3.4)\qquad \frac{\partial}{\partial\tau}u(x(\chi,\tau),\tau)$$

$$= u_t + \frac{dx}{d\tau}\cdot\nabla u = u_t + \beta\cdot\nabla u,$$

so that for example in the case $\epsilon = 0$, the equation (3.1) takes the simple form

$$(3.5a)\qquad \frac{\partial u}{\partial\tau}(x(\chi,\tau),\tau)$$

$$= f(x(\chi,\tau),\tau), \qquad \tau > 0,$$

$$(3.5b)\qquad u(x(\chi,0),0) = u_0(\chi),$$

which is a family of ordinary differential equations parametrized by $\chi\in\mathbb{R}^2$. This expresses the fact that in Lagrangean coordinates (χ,τ), the convection part $(u_t+\beta\cdot\nabla u)$ of (3.1a) takes a very simple form.

To be more precise, let us now introduce the mapping $F\colon Q\to Q$ defined by $(x,t) = F(\chi,\tau) = (x(\chi,\tau),\tau)$, where $x(\chi,\tau)$ satisfies (3.3). We recall that $|J| \equiv \det J$, where $J = \frac{\partial x}{\partial\chi}$ satisfies the equation $\frac{\partial|J|}{\partial\tau} = \mathrm{div}\,\beta\,|J| = 0$, $|J|(\cdot,0) = 1$ so that $|J|(\cdot,\tau) = 1$ for $\tau\geq 0$, which shows that $F\colon Q\to Q$ is 1-1 and onto. Defining now $\bar{u}(\chi,\tau) = u(x,t)$ where $(x,t) = F(\chi,\tau)$ and $\bar{\Delta}\bar{u}(\chi,\tau) = \Delta u(x,t)$, the problem (3.1) takes the following form in characteristic coordinates (χ,τ),

$$(3.6a)\qquad \bar{u}_\tau - \epsilon\bar{\Delta}\bar{u} = \bar{f} \qquad \text{in } Q,$$

$$(3.6b)\qquad \bar{u}(\cdot,0) = \bar{u}_0 \qquad \text{in } \mathbb{R}^2,$$

which is a variant of the standard heat equation with a variable coefficient analog $\bar{\Delta}$ of the usual Laplacian.

In the CSD-method for (3.1) the mapping F is now built in, in a suitable discrete form, which basically will make the CSD-method for (3.1) equivalent to the Discontinuous Galerkin method (the DG-method) for the parabolic problem (3.6) with tensor product elements in the characteristic coordinates (χ,τ). In

Eulerian coordinates (x,t) this will correspond to using space-time meshes on each slab S_n approximately oriented along characteristics. For the DG-method for parabolic problems with tensor product space-time elements, a very precise error analysis, including in particular integration over large time with frequent remeshing, is available ([J4], [J3]). Applying these results to (3.6) we obtain sharp error estimates for the CSD-method for (3.1), which significantly improve the corresponding estimates for the general SD-method without mesh orientation. In particular it follows that integration over large time and frequent remeshing is possible in the CSD-method without serious accumulation of errors or dissipation of fine scales.

3.2. The CSD-method

As above, let $0 = t_0 < t_1 < \dots < t_n < t_{N+1} = T$ a sequence of discrete time levels with associated time intervals $I_n = (t_n, t_{n+1})$, time steps $k_n = t_{n+1} - t_n$ and slabs $S_n = \mathbb{R}^2 \times I_n$, and let for each n a finite element space $W_n \subset H^1(\mathbb{R}^2)$ of piecewise linear functions on a mesh $T_n = \{\kappa\}$ on $\mathbb{R}^2$ with elements κ and mesh size $h_n = h_n(x)$ be given. Let us now for a given $q \geq 0$ for each n introduce a mapping $F_n: S_n \to S_n$ defined by

$$(3.7)\qquad (x,t) = F_n(\bar{x},\bar{t})$$

$$\equiv (\bar{x} + \sum_{j=0}^{q} \frac{(\bar{t}-t_n)^{j+1}}{j+1} B_j(\bar{x}),\ \bar{t}),$$

$$(\bar{x},\bar{t}) \in S_n,$$

where the $B_j \in W_n$ are functions to be defined so that

$$(3.8)\qquad B(x,t) \equiv \bar{B}(\bar{x},\bar{t})$$

$$\equiv \sum_{j=0}^{q} (\bar{t} - t_n)^j B_j(\bar{x}),$$

will be an approximation of $\beta(x,t)$ on S_n.

Here $(\bar{x},\bar{t})$ takes the role (locally on each S_n) of the characteristic coordinates (χ,τ) above. We note that with I the identity,

$$\frac{\partial x}{\partial \bar{x}}(\bar{x},\bar{t})$$

$$= I + \sum_{j=0}^{q} \frac{(\bar{t} - t_n)^{j+1}}{j+1} \nabla B_j(\bar{x}),$$

which proves that the mapping $F_n: S_n \to S_n$ is one-to-one and onto if

$$(3.9)\qquad \sum_{j=0}^{q} \frac{k_n^{j+1}}{j+1} \|\nabla B_j\|_{L_\infty(\mathbb{R}^2)} \leq c,$$

with c small enough. This puts a mild condition on the time step k_n if β is smooth. In the case of a non-smooth β, we may in order to avoid restrictions on the time step k_n, define the approximation B by first smoothing β. Next we note that $x = x(\bar{x},t)$ defined by (3.7) satisfies

$$(3.10a)\qquad \frac{dx}{dt} = B(x,t) \qquad t \in I_n,$$

$$(3.10b)\qquad x(\bar{x},t_n) = \bar{x}.$$

This means that the curves $x(\bar{x},t)$ defined by (3.10) will be approximations to the exact characteristics $x(\bar{x},t)$ defined by (3.3), if $B(x,t)$ approximates $\beta(x,t)$. In the simple basic case with $q = 0$, we have

$$(3.11)\qquad B(x,t) = \bar{B}(\bar{x},\bar{t}) = B_0(\bar{x}),$$

where $b_0 \in W_n$ will be an interpolant of $\beta^n = \beta(\cdot,t_n)$.

Continuing the notation used above, we adopt the convention of associating a function $v(x,t)$ on S_n with a given function $\bar{v}(\bar{x},\bar{t})$ on $\bar{S}_n$, and vice versa, by writing $v(x,t) = \bar{v}(\bar{x},\bar{t})$ where $(x,t) = F_n(\bar{x},\bar{t})$. We now introduce the following space of functions $v(x,t)$ with the corresponding $\bar{v}(\bar{x},\bar{t})$ being tensor-product (piecewise) polynomials in $(\bar{x},\bar{t})$:

$$V_n = \{v \in H^1(S_n): v(x,t) = \bar{v}(\bar{x},\bar{t})$$

$$= \sum_{j=0}^{q} (t - t_n)^j U_j, \text{ where } U_j \in W_n\},$$

in which an approximate solution U of (3.1) will be sought on each S_n. Note that $\{K: K = F_n(\kappa \times I_n),\ \kappa \in T_n\}$ gives a subdivision of S_n into oriented space-time elements ("titled curved prisms") $K = F_n(\kappa \times I_n)$, $\kappa \in T_n$, corresponding to the subdivision $\{\kappa \times I_n: \kappa \in T_n\}$ of $\bar{S}_n$ into straight prisms $\kappa \times I_n$.

We now define the CSD-method as follows: For $n = 0,1,\ldots$, find $U \equiv U|_{S_n} \in V_n$ such that

(3.12) $$\int_{S_n} (U_t + \beta\cdot\nabla U)(v + \delta(v_t + \beta\cdot\nabla v))dxdt$$

$$+ \int_{S_n} \hat{\epsilon}\nabla U\cdot\nabla v dxdt + \int_{\mathbb{R}^2} [U^n]v_+^n dx$$

$$= \int_{S_n} f(v + \delta(v_t + \beta\cdot\nabla v))dxdt$$

$$\forall v \in V_n,$$

where $U_-^0 = u_0$.

$$\delta = C_1(k_n^{-2} + h_n^{-2}|\beta - B|^2)^{-\frac{1}{2}} \text{ on } S_n,$$

$$\hat{\epsilon} = \max(\epsilon, C_2 hR(U)/|\nabla U|, C_3 h^{3/2}),$$

on S_n,

$$R(U) = \sum_{j=1}^{3} R_j(U),$$

$$R_1(U) = |U_t + \beta\cdot\nabla U - f - \mathrm{div}(\hat{\epsilon}\nabla U)| \text{ on } K,$$

$$R_2(U)|_K = \max_{\partial K} \frac{1}{2} |[\hat{\epsilon} n_x\cdot\nabla U]|/h_K,$$

$$R_3(U) = |[U^n]|/k_n \text{ on } S_n,$$

$[U^n]$ is now extended to S_n to be constant along the approximate characteristics $(x,t) = F_n(\bar{x},t)$, $t \in I_n$.

Let us now reformulate (3.12) in characteristic coordinates $(\bar{x},t)$. We shall then use the following notation where $(x,t) = F_n(\bar{x},t)$:

$$J_n = \frac{\partial x}{\partial \bar{x}},\quad \alpha = J_n^{-T}(\beta - B),\quad \bar{\nabla} = J_n^{-1}\nabla_{\bar{x}},$$

where $A^{-T} = (A^{-1})^T$ with T here denoting the transpose. We note that $J_n(x,t_n) = I$ for $x \in \mathbb{R}^2$ and recall that with $|J_n| = \det J_n$

(3.13) $$\frac{\partial}{\partial t}|J_n| = \mathrm{div}\, B|J_n|,\quad t \in I_n,$$

from which follows that

(3.14) $$\|J_n\| + \|J_n^{-1}\| \le C,$$

with $\|\cdot\|$ the matrix norm induced by the Euclidean norm in $\mathbb{R}^2$ if ∇B is bounded on S_n and k_n is small enough. We further note that

$$v_t + \beta\cdot\nabla v = v_t + B\cdot\nabla v + (\beta - B)\cdot\nabla v$$

$$= \frac{\partial}{\partial\bar{t}} v(x(\bar{x},\bar{t}), \bar{t}) + (\bar{\beta} - \bar{B})$$

$$\cdot \bar{J}_n^{-1} \nabla_{\bar{x}}\bar{v} = \bar{v}_{\bar{t}} + \bar{\alpha}\cdot\nabla_{\bar{x}}\bar{v} = \bar{v}_{\bar{t}} + \bar{\alpha}\cdot\nabla\bar{v},$$

where $\nabla_{\bar{x}}\bar{v} = \nabla\bar{v}$. Changing to $(\bar{x},\bar{t})$ coordinates, (3.10) now takes the form: For $n = 0,1,2,\dots N$ Find $\bar{U} \equiv \bar{U}|_{S_n} \in \bar{V}_n \equiv \{\bar{v}: v \in V_n\}$ such that

$$\int_{S_n} (\bar{U}_{\bar{t}}+\bar{\alpha}\cdot\nabla\bar{U})(\bar{v}+\delta(\bar{v}_{\bar{t}}+\bar{\alpha}\cdot\nabla\bar{v}))|J_n|d\bar{x}d\bar{t}$$

$$(3.15) \quad + \int_{S_n} \hat{\epsilon}\nabla\bar{U}\cdot\nabla\bar{v}|J_n|d\bar{x}d\bar{t} + \int_{\mathbb{R}^2} [U^n]v dx$$

$$= \int_{S_n} f(\bar{v}+\delta(\bar{v}_{\bar{t}} + \bar{\alpha}\cdot\nabla\bar{v}))|J_n|d\bar{x}d\bar{t}, \quad \forall\bar{v} \in \bar{V}_n,$$

with $\hat{\epsilon}$ and δ defined as in (3.12). Clearly, (3.15) corresponds to the DG-method with tensor product space-time elements for the following variant of (3.6):

$$(3.6) \qquad \bar{u}_{\bar{t}} + \bar{\alpha}\cdot\nabla\bar{u} - \mathrm{div}(\hat{\epsilon}\nabla u) = f,$$

where $\bar{\alpha}$ will be small if β is smooth. If $|\bar{\alpha}| \le C\hat{\epsilon}$, then (3.15) has full "parabolic nature" and the sharp error analysis from [EJ3] for the DG-method applies, whereas for $\bar{\alpha} = \mathcal{O}(1)$ the general SD-analysis may be used. Now, $|\bar{\alpha}| \le C|\bar{\beta} - \bar{B}|$, so the size of $\bar{\alpha}$ is directly connected to the quality of the velocity approximation B of β defined by (3.11). By standard interpolation error estimates with B a suitable interpolation approximation of β, we have

$$(3.17) \quad \|b - B\|_{L_\infty(Q)}$$

$$\le C(\|k^{q+1} D_\tau^{q+1}\beta\|_{L_\infty(Q)}$$

$$+ \|h^2D_\chi^2\beta\|_{L_\infty(Q)},$$

where $h = h_n$ and $k = k_n$ on S_n.

We now state error estimates for (3.15) under various assumptions on the size of $\bar{\alpha}$ as compared to $\hat{\epsilon}$. We recall that $\hat{\epsilon}$ will satisfy $C_3h^{3/2} \le \hat{\epsilon} \le C_2h$ if ϵ is small. As in [EJ7-8] we shall compare U with the solution $\hat{u}$ of (3.1) with ϵ replaced by $\hat{\epsilon}$. The error $u - \hat{u}$ changing ϵ to $\hat{\epsilon}$ in the continuous problem may be estimated separately, see [EJ7-8]. For simplicity of notation we identify $\hat{u}$ and u below.

<u>Case (i)</u>: $|\bar{\alpha}| \le C|\hat{\epsilon}|$ in S_n, $\forall n$.

As indicated, in this case we may directly use the sharp error estimates for the DG-method for parabolic problems given in [EJ3] under the (non-restrictive) assumption that

$$(3.18) \quad k_n \ge c \max_x h_n^2(x)/\hat{\epsilon}.$$

We then obta'n the following a priori error estimate assuming $\hat{\epsilon}$ and $\bar{\alpha}/\hat{\epsilon}$ are sufficiently smooth: There is a constant C such that for $t_N \le (\max\hat{\epsilon})^{-1}$ we have with $I = (0, t_N)$

$$(3.19) \quad \|u-U\|_{L_\infty(0,T;L_2)} \le$$

$$\le CL(\|k^{q+1}D_{\bar{t}}^{q+1}\bar{u}\|_{L_\infty(0,T;L_2)}$$

$$+ \|h^2D_{\bar{x}}^2\bar{u}\|_{L_\infty(0,T,L_2)}),$$

where $L = \max_{n \leq N} (\log \frac{t_{n+1}}{k_n} + 1)^{1/2}$, and $\|v\|_{L_\infty(0,T;L_2)} = \sup_{0 \leq t \leq T} \|v(t)\|_{L_2(\mathbb{R}^2)}$.

This is a fully optimal error estimate with the constant C independent of t_N and the time step k coupled with derivatives of $\bar{u}$ with respect to the characteristic coordinate $\bar{t}$. Improved variants with $q+1$ replaced by $2q+1$ may also be derived, see [EJ3]. Further, the following a posteriori analog of (3.19) was derived in [EJ3] (for simplicity stated in the case $f = 0$):

(3.20)
$$\|u-U\|_{L_\infty(0,T;L_2)} \leq CL(\|[U]\|_{L_\infty(0,T;L_2)} + \|h^2 D_h^2 U\|_{L_\infty(0,T;L_2)} + \|h^2[U]/k\|^*_{L_\infty(0,T,L_2)}),$$

where the $*$ indicates that the integrand should replaced by zero on I_n if $W_{n-1} \subset W_n$ and $[U]$ is extended to S_n as constant in time. We note the close similarity with (3.19).

The case $|\bar{\alpha}| \leq C\hat{\epsilon}$ considered typically would be relevant with $q = 0$ if $D_\tau \beta$ is small or with $q = 1$ if $k^2 \leq C\hat{\epsilon}$, assuming now $\frac{\partial^2 \beta}{\partial \tau^2}$ bounded, i.e., giving the time step restriction $k \leq Ch^{3/4}$ if $\hat{\epsilon} = Ch^{3/2}$. In both cases this indicates the possibility of choosing the time step k_n larger than the one corresponding to CFL > 1.

Case (ii): $|\bar{\alpha}| \leq C\,\hat{\epsilon}^{\frac{1}{2}}$.

In this case we state the following a posteriori derived in [EJ8]:

$$\|u-U\|_{L_2(Q)} \leq C\|(k_n + \hat{\epsilon}^{-1}h^2)R\|_{L_2(Q)},$$

with C independent of T. A typical case could now be as follows: $q = 0$, $\hat{\epsilon} = \mathcal{O}(h^{3/2})$, $k_n = Ch^{3/4}$, indicating again the possiblity of taking the time steps larger than the space steps corresponding to CFL > 1.

Case (iii): $|\bar{\alpha}| \geq C\,\hat{\epsilon}^{\frac{1}{2}}$.

In this case we may use the general estimate (2.3) for the SD-method (2.2) analogous to (1.16a).

3.3. Exact transport and projection ETP

In the case β = constant, $q = 0$, $B = \beta$, in which case we may take $\delta = 0$, the CSD method (3.12) takes the form (assuming also $f = 0$ for simplicity): Find $U \in V_n = \{v\colon v(x,t) = w(\bar{x})$ with $x = \bar{x}+(t-t_n)\beta$, $w \in W_n\}$ such that

(3.21)
$$\int_{S_n} (U_t+\beta\cdot\nabla U)v\,dxdt + \int_{S_n} \hat{\epsilon}\nabla U\cdot\nabla v\,dxdt + \int_{\mathbb{R}^2} [U^n]v_+^n\,dx = 0, \quad \forall v \in V_n.$$

Since $v_t + \beta\cdot\nabla v = 0$ if $v \in V_n$, we can write (3.21) as follows: Find $U_+^n \in W_n$ such that

(3.22)
$$\int_{\mathbb{R}^2} U_+^n v\,dx + k_n \int_{\mathbb{R}^2} \hat{\epsilon}\nabla U_+^n\cdot\nabla v\,dx = \int_{\mathbb{R}^2} U_+^{n-1}(x-k_n\beta)v(x)dx, \quad \forall v \in W_n,$$

where we used the fact that by the definition of V_{n-1}, $U_-^n(x) =$

$U_+^{n-1}(x-k_{n-1}\beta)$. With $\hat{\epsilon} = 0$ this means that

(3.23) $\quad U_+^n = P_n T_n U_+^{n-1},$

where $P_n: L_2(\mathbb{R}^2) \to V_n$ is the L_2-projection defined by $(P_n w,v) = (w,v)$ $\forall v \in V_n$, and T_n is the translation defined by

$$(T_n v)(x) = v(x - k_{n-1}\beta).$$

Clearly, (3.22) corresponds to a method for the problem (3.1) with $\epsilon = 0$ based on "exact transport + L_2-projection". With the proper definition of $\hat{\epsilon}$ in the present case, i.e.,

$$\hat{\epsilon} = C_2 h^2 |[U^n]|/k_n,$$

the CSD-method (3.22) corresponds to a method of again the form (3.23) with now a modified L_2-projection $\tilde{P}_n: H^1(\mathbb{R}^2) \to V_n$ defined by

(3.24) $\quad (\tilde{P}_n w,v) + (\hat{\epsilon} k_n \nabla P_n w, \nabla v) = (w,v) \qquad \forall v \in V_n,$

where $\hat{\epsilon} k_n = C_2 h^2 |P_n w - w|$. In the presence of discontinuities the modified projection $\tilde{P}_n$ shows improved performance as compared to the standard L_2-projection P_n, with monotone resolution of discontinuities where P_n gives mild oscillations.

We have now seen that the CSD-method in the case β constant $\epsilon = f = 0$, and $q = 0$ reduces to a method of the form "exact transport + projection", or ETP for short, with a built-in modified L_2-projection with good stability properties. More generally, the CSD-method will reduce to a method of this form (in the case $\epsilon = 0$) if the discrete velocity B in the CSD-method defined by (3.8) is equal to the exact velocity β. We recall that the idea of ETP underlines the Godunov method and generalizations thereof (VanLeer, PPM), and can probably be viewed, together with the idea of centered difference approximations and artificial viscosity, as the bearing principle in classical CFD not including finite element methods.

Although thus ETP has been fairly successful as a principle for the generation of schemes in CFD, ETP does not offer a full discretization procedure since it requires the "exact transport" step to be performed form one discrete time level to the next, which in general is highly non-trivial (e.g. requiring the solution of Riemann problems).

Now, our point is that the CSD-method, which coincides with ETP in simple cases, and which gives a full discretization for general problems, may be viewed as the natural generalization of ETP. The following advantages are obtained this way:

(i) Fully discrete schemes for general problems are obtained; it is no longer necessary to solve the given equations exactly between discrete time levels.

(ii) The analysis of the CSD-method is more precise than the classical analysis of ETP. For example the CSD-analysis shows accuracy $\mathcal{O}(h^2/\sqrt{k})$ with piecewise linears in space and $q = 0$ when k is the time step, while the ETP-analysis gives $\mathcal{O}(h^2/k)$. Further, with the sharp CSD-analysis related to (3.19) it can be proved that in the CSD-method a discontinuity may be propagated over long time intervals with little smearing, see [J4].

4. THE SD AND CSD-METHODS FOR THE INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

4.1. Formulation of the SD and CSD-methods

In this section we present the SD and CSD-methods for the incompressible Navier-Stokes equations in $\mathbb{R}^d$ $(d = 2,3)$:

Find the velocity $u = (u_i)_{i=1}^d$, and the pressure p such that

(4.1a) $u_t + (u\cdot\nabla)u + \nabla p - \epsilon\Delta u = f$ in Q,

(4.1b) $u = 0$ on $\Gamma \times (0,T)$

(4.1c) $u(\cdot,0) = u_0$,

where $Q = \Omega \times (0,T)$, and Ω is a bounded domain in $\mathbb{R}^d$, ϵ is a positive (small) constant and f and u_0 are given data.

Let now as above $\{t_n\}$ be a sequence of discrete time steps and let for each n, $W_n \subset H_0^1(\Omega)$ be a finite element space consisting of piecewise linear functions on a mesh $T_n = \{\kappa\}$ of mesh size h_n. For a given velocity field U on $S_n = \Omega \times I_n$ let us now introduce the particle paths $x(\bar{x},\bar{t})$ defined by

(4.2a) $\dfrac{dx}{d\bar{t}} = U(x,\bar{t}) \quad \bar{t} \in I_n$,

(4.2b) $x(\bar{x}, t_n) = \bar{x}, \quad \bar{x} \in \Omega$,

and the corresponding mapping $F_n^U: S_n \to S_n$ defined by $(x,t) = F_n^U(\bar{x},\bar{t})$ where $x = x(\bar{x},\bar{t})$ satisfies (4.2). We next introduce for a given $q \geq 0$, the spaces

(4.3a) $\bar{V}_n^U = \{\bar{v} \in H^1(S_n)^d: \bar{v}(\bar{x},\bar{t})$

$= \sum_{j=0}^{q} (\bar{t}-t_n)^j U_j(\bar{x}),\ U_j \in [W_n]^d\}$,

(4.3b) $\bar{Q}_n^U = \{\bar{q} \in H^1(S_n): \bar{q}(\bar{x},\bar{t})$

$= \sum_{j=0}^{q} (\bar{t}_n-t_n)^j q_j(\bar{x}),\ q_j \in W_n\}$,

together with their analogs in (x,t)-coordinates:

(4.3c) $V_n^U = \{v: \bar{v} \in \bar{V}_n^N\}$,

$Q_n^U = \{q: \bar{q} \in \bar{Q}_n^U\}$.

We can now formulate the CSD-method, for simplicity without shock-capturing ($\hat{\epsilon} = \epsilon$), as follows: For $n = 0,1,2,\ldots, N$, find $(U,P) = (U,P)|_{I_n} \in V_n^U \times W_n^U$, such that

(4.4) $(U_t + (U\cdot\nabla)\, U, v)_n - (P, \operatorname{div} v)_n$

$+ (q, \operatorname{div} U)_n + \epsilon(\nabla U, \nabla v)_n$

$+ \delta_1(U_t + (U\cdot\nabla)U + \nabla P - \epsilon\Delta U,$

$v_t + (U\cdot\nabla)v + \nabla q - \epsilon\Delta v)_n$

$+ (\delta_2 \operatorname{div} U, \operatorname{div} v)_n + (\delta_3 U, v)_n$

$= (f, v + \delta_1(v_t + (U\cdot\nabla)v$

$+ \nabla P - \epsilon\Delta U))_n \quad \forall (v,q) \in V_n^U \times Q_n^U$,

where

$\delta_1 = C_1 k_n$,

$$\delta_2 = \delta_2(\operatorname{div} U) = \begin{cases} C_2 h & \text{if } \operatorname{div} U \leq K \\ C_2 & \text{if } \operatorname{div} U > K \end{cases}$$

$$\delta_3 = \delta_3(\operatorname{div} U) = \begin{cases} 0 & \text{if } \operatorname{div} U \leq K \\ \operatorname{div} U & \text{if } \operatorname{div} U > K, \end{cases}$$

and as above

$$(v,w)_n = \int_{I_n} (v,w)dt,\ (v,w) = \int_\Omega v\cdot w dx,$$

$$(\nabla v,\nabla w)_n = \int_{I_n} (\nabla v,\nabla w)dt,$$

$$(\nabla v,\nabla w) = \sum_{i=1}^{d} \int_\Omega (\nabla v_i \cdot \nabla w_i\, dx$$

Notice that as defined the CSD-method (4.4) involves an additional non-linearity since V_n^U depends on U. By standard fixed point arguments it can be proved that (4.3) has at least one solution (U,P). Normally we expect to be able to find such a solution by computing a sequence of solutions $(U^{(k)}, P^{(k)})$ by seeking $(U^{(k)}, P^{(k)}) \in V_n^{U^{(k-1)}} \times Q_n^{U^{(k-1)}}$. Depending on the choice of initial approximation $(U^{(0)}, P^{(0)})$ and the number of iterations we may get different variants of the CSD-method: For example with $U^{(0)} = 0$ and $k = 1$ we get a standard SD-method with non-oriented tensor-product elements in (x,t). In a typical implementation of the CSD-method we would let $U^{(0)}$ be obtained from the previous time step and iterate once ($k = 1$).

This being said, let us now consider the CSD-method in the full non-linear formulation (4.4). We note that with (4.2) satisfied, we have

$$\frac{\partial \bar{U}}{\partial \bar{t}} \equiv \frac{\partial}{\partial \bar{t}} U(x(\bar{x},\bar{t}),\bar{t}) = U_t + U\cdot\nabla U,$$

which shows that in CSD-method in the local coordinates $(\bar{x},\bar{t})$ on each slab S_n, the convection term takes a very simple form. In particular, we have that when written in $(\bar{x},\bar{t})$-coordinates, the discrete equations (4.3) correspond to a modified Stokes problem.

4.2 An a posteriori error estimate for the CSD-method

Let us now give a (formal) proof of an a posteriori error estimate for the CSD-method. The proof is based on a strong stability estimate for a linearized dual problem, which appears to be fundamental but seems difficult to establish by analytical techniques in interesting case . However, there is a clear possibility of testing this stability condition numerically.

For simplicity, let us consider the method (1.3) with $\delta_j = 0$, $j = 1,2,3$, which does not essentially affect the details of the argument to follow. Let us introduce the following linearized dual problem: Find $(\varphi,\theta) \in L_2(I; [H_0^1(\Omega)]^d \times L_2(\Omega)) \equiv W$ such that in Q

$$\text{(4.5a)} \quad -\varphi_t - (u\cdot\nabla)\varphi + \nabla U\cdot\varphi + \nabla\theta - \epsilon\Delta\varphi = e$$

$$\text{(4.5b)} \qquad \operatorname{div}\ \varphi = 0$$

$$\left.\begin{matrix}\text{(4.5c)}\\ \text{(4.5d)}\end{matrix}\right\} \qquad \begin{matrix}\operatorname{div}\ \varphi = 0 \text{ on } \Gamma\times I,\\ \varphi(\cdot,T) = 0 \text{ in } \Omega,\end{matrix}$$

where $e = u - U$. Multiplying (4.4) by e and integrating over I_n together with integration by parts gives

$$\text{(4.6)} \quad \|e\|^2_{L_2(Q)}$$

$$= \sum_{n=0}^{N} \{(-\varphi_t - (u\cdot\nabla)\varphi + \nabla U\cdot\varphi, e)_n + (\nabla\theta, e)_n + (\epsilon\nabla\varphi, \nabla e)_n\}$$

$$= \sum_{n=0}^{N} \{(\varphi,e_t)_n + (u\cdot\nabla e,\varphi)_n + (\nabla U\cdot\varphi,e)_n - (\theta,\operatorname{div} e)_n + (\epsilon\nabla\varphi,\nabla e)_n - (p-P,\operatorname{div}\varphi)_n\}$$

$$+ \sum_{n=0}^{N} \int_\Omega ([U^n], \varphi^n)dx$$

$$= \sum_{n=0}^{N} \{(u_t + u\cdot\nabla u + \nabla p, \varphi)_n + (\epsilon\nabla u, \nabla\varphi)_n - (U_t + U\cdot\nabla U + \nabla P, \varphi)_n + (\epsilon\nabla U, \nabla\varphi)_n + (\theta, \operatorname{div} U)_n\} + \sum_{n=0}^{N} \int_\Omega ([U^n], \varphi^n)dx$$

$$= -\sum_{n=0}^{N} \{(U_t + U\cdot\nabla U + \nabla P - f, \varphi-\Phi)_n + (\epsilon\nabla U, \nabla(\varphi-\Phi))_n + (\operatorname{div} U, \theta-\Theta)_n\}_n$$

$$+ \sum_{n=0}^{N} \int_{\Omega} ([U^n], \varphi^n - \Phi^n)dx,$$

where $(\Phi, \Theta)|_{S_n} \in V_n^U \times Q_n^U$ will be chosen to interpolate (φ,Θ).

We shall now assume the following stability estimate for the linear dual problem (4.5): There is a constant C such that

(4.7) $$\|\varphi_t + (u\cdot\nabla)\varphi\|_{L_2(Q)} + \|\nabla\theta - \epsilon\Delta\varphi\|_{L_2(Q)} + \|\epsilon^{\frac{1}{2}}\nabla\varphi\|_{L_2(Q)} + \|\theta\|_{L_2(Q)} \le C\|e\|_{L_2(Q)}.$$

Without the term $\nabla U\cdot\varphi$ present this estimate follows by multiplying (4.4a) by $(\varphi_t + (u\cdot\nabla)\varphi)$ and integrating by parts using the fact that $\operatorname{div} u = \operatorname{div}\varphi = 0$. Note that by elliptic regularity for the Stokes equations it follows that

(4.8) $$\|\nabla\theta\|_{L_2(\Omega)} + \|\epsilon D^2\varphi\|_{L_2(\Omega)} \le C\|\nabla\theta - \epsilon\Delta\varphi\|_{L_2(\Omega)}.$$

Combining (4.5)-(4.7) and recalling (1.8) we obtain the following (formal) a posteriori error estimate

(4.9) $$\|e\|_{L_2(Q)} \le C\Big[\|h^2\epsilon^{-1}R\|_{L_2(Q)} + \|k\epsilon^{-\frac{1}{2}}R\|_{L_\infty(Q)} + \|\operatorname{div} U\|_{L_2(Q)}\Big],$$

where

$$R = |U_t + U\cdot\nabla U + \nabla P - f| + \frac{|[U^n]|}{k_n},$$

and for simplicity the usual jump terms involving $\epsilon\frac{\partial u}{\partial n}$, have been omitted. Further, the viscosity coefficient ϵ in (4.9) should be replaced by an artificial viscosity $\hat\epsilon$ defined as above. With suitable choice of $\hat\epsilon$, the estimate (4.9) appears to give a reasonably efficient adaptive algorithm.

The stability properties of the dual problem (4.5) obviously play a crucial role. In general the stability of (4.5) can only be evaluated computationally. We have here very briefly scratched the surface of a topic we hope to develop further in the future: Adaptive (C)SD-methods for incompressible flow, with a very large area of application.

4.3 The CSD-method for free boundary flow

The CSD-method is ideally suited to handle flow problem with free boundaries or moving boundaries with prescribed motion: Just let the nodes on the boundary move according to (4.2) with U a computed or prescribed velocity. To be more precise, let us present the CSD-method for the Navier-Stokes equations with free boundary occupying the volume $\Omega(t)$ at time $t \in (0,T)$: Find (u,p) such that for $t \in I$,

(4.10a) $u_t+(u\cdot\nabla)u+\nabla p - \operatorname{div}\sigma=f$ in $\Omega(t)$,

(4.10b) $\operatorname{div} u=0$ in $\Omega(t)$,

(4.10c) $\sigma\cdot n=0$ on $\Gamma(t)$,

(4.10d) $u(\cdot,0)=u_0$ in $\Omega(0)$,

where $\sigma = \{\sigma_{ij}\}$ is the stress tensor defined by

$$\sigma_{ij} = -p\delta_{ij} + 2\mu\epsilon_{ij}(u),$$

$$\epsilon_{ij}(u) = \frac{1}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right],$$

and $\sigma\cdot n = (\sum_j \sigma_{ij}n_j)$ is the stress on the boundary $\Gamma(t)$ of $\Omega(t)$ with outward normal n. Here $\Omega(t) = \{x(\chi,t): \chi \in \Omega(0)\}$ where $x(\chi,t)$ satisfies

$$\begin{cases} \frac{dx}{dt} = u(x,t), & t > 0. \quad (4.11a)\\ x(\chi,0) = \chi. & (4.11b)\end{cases}$$

Note that (4.10c) states that the (total) stress on the free boundary is zero. Under sufficient regularity assumptions it is known that (4.10) admits a unique solution if T is small enough.

To define the CSD-method for (4.109) let $\{t_n\}$ be a squence of discrete time levels, suppose the approximation Ω_n of $\Omega(t_n)$ is given, let W_n be the space of continuous piecewise linears on a triangulation T_n of Ω_n and define V_n^U and Q_n^U by (4.3). Note that in this case the functions in W_n are not restricted to be zero on the boundary of Ω_n. Note further that in this case the functions in V_n^U and $Q=_n^U$ are defined on $\tilde{S}_n = F_n^U(S_n)$ with $S_n = \Omega_n \times I_n$. The CSD-method can now be formulated as follows ([Ha3]): Find $(U,P) \in V_n^U \times Q_n^U$ such that

(4.12) $(U_t + (U\cdot\nabla U)U,v)_n$

$- (P,\operatorname{div} v)_n + (q,\operatorname{div} U)_n + 2\mu(\epsilon(U), \epsilon(v))_n$

$+ \delta_1(U_t + (U\cdot\nabla)U + \nabla P, v_t + (U\cdot\nabla)v + \nabla q)_n$

$+ (\delta_2 \operatorname{div} U, \operatorname{div} v)_n + (\delta_3 U,v)_n +$

$\delta_3 \langle(2\mu\epsilon(U) - pI)\cdot n, (2\mu\epsilon(v) - qI)\cdot n\rangle_n$

$= (f, v + \delta_1(v_t + (U\cdot\nabla)v + \nabla q))_n$

$\forall (v,q) \in V_n^U \times Q_n^U,$

where δ_j, $j = 1,2,3$, are defined as above, δ_4 is a positive constant independent of h, and

$$(v,w)_n = \int_{I_n} \int_{\Omega_n(t)} v\cdot w\,dx\,dt$$

$$= \int_{\tilde{S}_n} v\cdot w\,dx dt,$$

$$\langle v,w\rangle_n = \int_{I_n} \int_{\partial\Omega_n(t)} v\cdot w\,ds\,dt,$$

where $\Omega_n(t) = \{x(\chi,t): \chi \in \Omega_n\}$ with x satisfying (4.2) for $\chi \in \Omega_n$. We note that boundary condition (4.10c) is enforced weakly in (4.12) since the velocities in V_n^U are not restrained on the boundary of $\tilde{S}_n = F_n^U(S_n)$. Further, we note the S_4-term gives least squares control of the discrete boundary stress.

In Section 8, we present some numerical results from [Ha3] for the method (4.12) applied to non-stationary fountain flow using $q = 0$, $k = 1$ and with $U^{(0)}|_{S_n}$ given by $U|_{S_{n-1}}$.

5. SD-METHODS FOR COMPRESSIBLE FLOW

5.1 Formulation of the SD-method

In this section we present the SD-method for the compressible Euler equations for a perfect gas in $\mathbb{R}^2$:

(5.1a) $u_t + \sum_{i=1}^{2} f_i(u)_{x_i} = 0 \qquad x \in \mathbb{R}^2,\ t > 0,$

(5.1b) $\qquad u(x,0) = u_0(x) \quad x \in \mathbb{R}^2,$

where

$$u = \rho \begin{bmatrix} 1 \\ w_1 \\ w_2 \\ e \end{bmatrix}, \quad f_i = w_i u + p \begin{bmatrix} 0 \\ \delta_{1i} \\ \delta_{2i} \\ w_i \end{bmatrix}.$$

Here ρ is the density, $w = (w_1, w_2)$ is the velocity, e is the total energy density, $p = (\gamma-1)(\rho e - \rho(w_1^2 + w_2^2)/2)$ is the pressure, δ_{ij} the Kronecker delta, and $\gamma > 1$ is a constant. The conservation law (5.1) can also be written in the form

$$(5.2)\qquad u_t + \sum_{i=1}^{2} A_i(u)u_{x_i} = 0,$$

where $A_i = \frac{\partial f_i}{\partial u}$ is the Jacobian of $f_i(u)$.

Let now $\{S_n\}$ be the usual sequence of space-time slabs $S_n = \mathbb{R}^2 \times I_n$, let for each n $V_n \subset H^1(S_n)^4$ be a finite element space and set $V = \prod_{n\geq 0} V_n$. The general SD-method for (5.1) can now be formulated as follows: Find $U \in V$ such that $U \equiv U|_{S_n} \in V_n$ satisfies:

$$(5.3)\qquad (U_t + \sum_i A_i(U)U_{x_i},$$
$$v + \delta(v_t + \sum_i A_i^T(U)v_{x_i}))_n + ([U^n], v_+^n)$$
$$+ (\hat{\epsilon}\nabla U, \nabla v)_n + (\hat{\epsilon}U_t, v_t)_n = 0 \quad \forall v \in V_n,$$

where T again denotes transpose and

$$(5.4)\qquad \delta^T = \delta(U)^T$$
$$= \frac{1}{2}(k_n^{-2}I + h_n^{-2}\sum_{i=1}^{2} A_i(U)^2)^{-\frac{1}{2}} \text{ on } S_n,$$

$$(5.5)\qquad \hat{\epsilon} = \hat{\epsilon}(U) = C_2 h \frac{|R(U)|}{|\nabla U| + h},$$
$$R(U) = |U_t + \sum_i A_i(U)U_{x_i}|$$
$$+ |[U^n]|/k_n \quad \text{on } S_n.$$

The simplest instance of the SD-method (5.3) is obtained with a $\mathbb{P}_1 \times \mathbb{P}_0$-approximation in space-time on tensor product elements $K = \tau \times I_n$ (or "tilted" such elements) with the basis functions continuous in x and discontinuous in time, see Section 5.3 below.

Remark 5.1. To see that square root in (5.4) is well defined we recall ([H1]) that there is a positive definite matrix $A_0 = A_0(U)$ such that $\bar{A}_i \equiv A_i A_0$ is symmetric, i = 1,2,. Thus

$$A_0^{-\frac{1}{2}} A_i A_0^{\frac{1}{2}} = A_0^{-\frac{1}{2}} \bar{A}_i A_0^{-\frac{1}{2}}$$

is symmetric, so the similarity transform induced by $A_0^{\frac{1}{2}}$ transforms the matrix $M \equiv (k_n^{-2} I + h^{-2} \sum_i A_i^2)$ to an obviously positive definite symmetric matrix. It follows that M has positive eigenvalues and a full set of eigenvectors which shows that $M^{-\frac{1}{2}}$ can be computed. In [Ha4] explicit formulas for the eigenvalues and eigenvectors of M are given. We recall that $A_0 = \eta_{uu}^{-1}$ where $\eta(u) = - c\rho \log(p\rho^{-\gamma})$ is the entropy. ■

5.2. Entropy consistency

To prove convergence of any numerical method for the Euler equations (5.1) is for the moment impossible, since existence of solutions of the Euler equations has not been proved mathematically. For the SD-method the following weaker result is possible to prove (with polynomial approximation in space-time of any order): Limits of SD-solutions will satisfy any entropy condition for (5.1)([JSzH], [Sz1]). A corresponding result in the same generality for finite difference/volume methods is known only for first order approximations.

5.3. An explicit form of the SD-method

The SD-method (5.3) leads to a system of non-linear equations to solve for each slab S_n. If the time step is sufficiently small (corresponding to CFL < 1/2 say), we expect to be able to solve this system with few Newton-like iterations. We shall now consider the simplest case of $\mathbb{P}_1 \times \mathbb{P}_0$-approximation in (x,t) on tensor product elements $\tau \times I_n$ (or more generally "tilted" such elements), continuous in x and discontinuous in t. Writing $U^n \equiv U|_{S_n}$ the SD-method (5.3) can in this case be formulated as follows: Find $U^n \in W_n$ such that

$$(5.6) \quad (U^n - U^{n-1}, v) + (\sum_i A_i(U^n)U^n_{x_i}, v + \delta(\sum_i A_i(U^n)^T v_{x_i}))k_n + (\hat{\epsilon}\nabla U^n, \nabla v)k_n = 0 \qquad \forall v \in W_n,$$

where $W_n \subset H^1(\mathbb{R}^2)^4$ consists of continuous piecewise linears and the corresponding $V_n \subset H^1(S_n)^4$ is defined by $V_n = \{v \in H^1(S_n)^4 : v(\cdot,t) = w \text{ for } t \in I_n, \text{ where } w \in W_n\}$.

To compute U^n from (5.6), we may iterate in various ways: The simplest possible variant is obtained by lumping the mass matrix related to the inner product $(\cdot,\cdot)$ in the first term, and changing the index n to $n-1$ in all terms except the first, which gives a fully explicit scheme of the form ([Ha 4])

$$(5.7) \quad U^n_j = U^{n-1}_j - k_n(\sum_i f_i(U^{n-1})_{x_i}, \varphi_j) - k_n(\sum_i A_i(U^{n-1})U_{x_i}, \delta\sum_i A_i(U^{n-1})^T \varphi_{j,x_i}) - k_n(\hat{\epsilon}\nabla U^{n-1}, \nabla\varphi_j),$$

where $U^n = \sum_j U^n_j \varphi_j(x)$ with $\{\varphi_j\}$ the standard $\mathbb{P}_1$-basis functions of W_n. The resulting method may be viewed as a variant of certain well-known "upwind" finite difference methods, see [Ha4]. Many variants of (5.7) with improved performance in certain cases may be obtained by keeping the mass matrix and e.g. the $\hat{\epsilon}$-term on the left hand side of (5.6) in an iterative method, and using e.g. the conjugate gradient method, at each iterative step.

In Section 8 we present some numerical results for the explicit method (5.7) applied to a standard test probelm: Mach 3 flow in a channel with a step up. We further give some results for (5.6) extended to the compressible Navier-Stokes equations for flow over a flat plate. In both cases the adaptive mesh control is based on (formal) a posteriori error estimates of the form (0.10).

6. The DG-method for conservation laws

In this section we present the Discontinuous Galerkin method (DG-method) which is a variant of the SD-method with discontinuous approximation in space as well as in time. The DG-method may be viewed as a generalized Finite Volume method. The DG-method is fully analogous to the SD-method with the basic two modifications (0.1) and (0.2) and in addition certain jump terms related to inter-element discontinuities.

Let as bove $\{t_n\}$ be a sequence of discrete time levels, $I_n = (t_n, t_{n+1})$, $S_n = \mathbb{R}^2 \times I_n$, and let $T_n = \{K\}$ be a finite element triangulation of S_n into space-time elements K of diameter h_K. Typically the elements K may be prisms $\kappa \times I_n$ with κ a triangle or "tilted" such prisms, or tetrahedrons. Define for $q \geq 0$

$$V_n = \{v \in L_2(S_n) : v|_K \in \mathbb{P}_q(K), K \in T_n\},$$

$$V = \prod_{n \geq 0} V_n.$$

We shall present the DG-method for a scalar conservation law:

$$(6.1a) \quad u_t + \sum_{i=1}^{2} f_i(u)_{x_i} = 0 \quad \text{in } \mathbb{R}^2 \times \mathbb{R}_+,$$

$$(6.1b) \quad u(\cdot,0) = x_0 \quad \text{in } \mathbb{R}^2,$$

where the $f_i: \mathbb{R} \to \mathbb{R}$ are smooth fluxes and $u_0 \in L_\infty(\mathbb{R}^2)$ has compact support. The DG-method for (6.1) reads as follows: Find $U \in V$ such that $U \equiv U|_{S_n}$ satisfies

(6.2) $$\sum_{K\in T_n} \{\int_K (U_t + \sum f_i(U)_{x_i})$$

$$(v + \delta(v_t + \sum_i f_i'(U)v_{x_i}))dxdt$$

$$+ \int_K \hat{\epsilon}\, \hat{\nabla}U\cdot\hat{\nabla}v\, dxdt$$

$$\int_{\partial K}(F_K(U) - f(U)\cdot n_K)v_K ds\} = 0$$

$$\forall v \in V_n,$$

where n_K is the outward unit normal to K, $\hat{\nabla}v = (\nabla v, v_t)$,

$$v_K = v|_K,$$

$$f(U) = (U, f_1(U), f_2(U)),$$

(6.3) $$F_K(U) = \frac{1}{2}(f(U_K) + f(U_K))\cdot n_K$$

$$+ C_K(U_K - U_{K'}) \text{ at } (x,t) \in \partial K,$$

where K' is an element sharing the face S common to K and K' containing (x,t), and

$$C_K = \frac{1}{2} \quad \text{if } n_K = \pm(1,0,0),$$

$$C_K = Ch^{-\gamma} \quad \text{otherwise,}$$

with C a positive constant and $\gamma > 0$ small. Further,

$$\delta = C_1(k_n^{-2} + h_n^{-2}\sum_i f_i'(U)^2)^{-\frac{1}{2}}$$

$$\hat{\epsilon} = C_2 h^\alpha(|U_t + \sum_i f_i(U)_{x_i}|$$

$$+ \max_{S\subset\partial K} |U_K - U_{K'}|/h_K \qquad \text{in } K,$$

with $\frac{3}{2} < \alpha < 2$. Note that here we use a variant of the artificial viscosity $\hat{\epsilon}$ used above.

Remark. Note that if $n_K = (-1,0,0)$ corresponding to the "bottom" of K, then

$$F_K(U) = -\frac{1}{2}(U_+^n + U_-^n)$$

$$+\frac{1}{2}(U_+^n - U_-^n) = -U_-^n$$

where as above $U_\pm^n = \lim_{s\to 0^+} U(t_n \pm s)$, so that

$$((F(U_K) - f(U_{K'}))\cdot n_K)v_K$$

$$= (U_+^n - U_-^n)v_+^n,$$

which gives the usual jump term. Further, if $n_K = (1,0,0)$, then

$$(F_K(U) - f(U_K))\cdot n_K \equiv 0,$$

which means that there is no coupling forward in time. Note further that the F_K are basically the Lax-Friedrichs' fluxes.

For $q = 0$, (6.2) gives an implicit variant of the Finite Volume method. The accuracy of (6.2) is formally of order $\mathcal{O}(h^{q+\frac{1}{2}})$, thus of accuracy higher than one if $q \geq 1$. In particular, the use of the Lax-Friedricks' fluxes (6.3) does not degrade the accuracy to first order if $q \geq 1$.

It is possible to prove convergence of the method (6.2) with $q \geq 0$ using the same method of proof as for the SD-method with continuous basic functions in space based on the DiPerna uniqueness result for measure valued solutions of scalar conservation laws (see [Sz2], [JJ]). ■

7. ADAPTIVE CSD-METHOD FOR BURGERS' EQUATION

7.1. Introduction

In [EJ7-8], [J2], [J5] and [JSz2], we prove *a posteriori* error estimates and formulate corresponding adaptive algorithms for linear convection-diffusion problems, the linear wave equation, and systems of conservation laws in one space dimension, respectively. Further, in [HJ] we discuss extensions and present computational results for compressible flow in two space dimensions. As far as we know, our results are the first to show that reliable and efficient adaptive error control based on *a posteriori* error estimates is possible for finite element methods for hyperbolic problems. Of particular interest are the results on adaptive finite element methods for systems of conservation laws, which seem to open a large area of application. In the proofs of the *a posteriori* error estimates we use our new strong stability concept in a crucial way; using classical stability concepts it appears to be impossible to obtain useful *a posteriori* error stimulates for hyperbolic problems. The case of conservation laws presents a remarkable example of the utility of the new concept of strong stability: In this case the linearized dual problem is indeed strongly stable, but is unstable with the classical weak stability concept corresponding to the fact that solutions of conservation laws do not in general show continuous dependence L_2 with respect to change of initial data or right hand side in L_2.

To illustrate the essential points, we shall in this section indicate the proof of an *a posteriori* error estimate for a finite element method for a scalar conservation law in one space dimension (Burgers' equation). For more details and the important extension to the case of systems, we refer to [JSz2].

7.2. An *a posteriori* error estimate for a finite element method for Burgers' equation

We consider Burgers' equation: Find the scalar function $u^\epsilon = u^\epsilon(x,t)$ such that for $x \in \mathbb{R}$, $t > 0$

$$\text{(7.1a)} \quad \frac{\partial u^\epsilon}{\partial t} + \frac{\partial}{\partial x}\left[\frac{1}{2}(u^\epsilon)^2\right] - \epsilon\frac{\partial^2 u^\epsilon}{\partial x^2} = 0,$$

$$\text{(7.1b)} \qquad u^\epsilon(x,t) \to 0, \ x \to \pm\infty, \ t > 0,$$

$$\text{(7.1c)} \qquad u^\epsilon(x,0) = u_0(x), \quad x \in \mathbb{R},$$

where $u_0 \in L_\infty(\mathbb{R})$ has compact support, and ϵ is a small positive constant. For $\epsilon > 0$ this problem is uniquely solvable. As ϵ tends to zero, the solution u^ϵ will converge to a limit u, the *entropy solution* of the inviscid Burgers' equation corresponding to (7.1) with $\epsilon = 0$. Even if the data u_0 is smooth, $u(\cdot,t)$ may become discontinuous in finite time, corresponding to the development of shocks. The entropy condition states that at shocks (discontinuities), we have $u^-(x,t) > u^+(x,t)$, where

$$u^\pm(x,t) = \lim_{s \to 0^\pm} u(x+s,t)$$

is the left-hand and right-hand limit, respectively. This means that close to shocks $\frac{\partial u^\epsilon}{\partial x}$ may be very large negative (but $\frac{\partial u^\epsilon}{\partial x}$ will be bounded above). In such a case the linearized problem

$$\text{(7.2a)} \quad \frac{\partial \phi}{\partial t} + \frac{\partial}{\partial x}(u^\epsilon \phi) - \epsilon\frac{\partial^2 \phi}{\partial x^2} = 0, \text{ in } Q,$$

$$\text{(7.2b)} \qquad \phi(x,t) \to 0, \ x \to \infty$$

$$\text{(7.2c)} \qquad \phi(x,0) = \phi_0(x) \ x \in \mathbb{R},$$

where $Q = \mathbb{R} \times \mathbb{R}_+$ obtained by linearizing (7.1a) around the solution u^ϵ, is unstable in L_2, since multiplicating (7.2a) with ϕ and integrating with respect to x leads to

$$\frac{1}{2}\frac{d}{dt}\int_{\mathbb{R}} \phi^2(x,t)dx$$

$$+ \epsilon \int_{\mathbb{R}} \left[\frac{\partial \phi(x,t)}{\partial x}\right]^2 dx = -\frac{1}{2}\int_{\mathbb{R}} \frac{\partial u^\epsilon}{\partial x}\phi^2(x,t)dx,$$

where the right-hand side may be large if $\frac{\partial u^\epsilon}{\partial x}$ is large negative, which corresponds to the fact that (7.1) is not continuous in L_2 with respect to L_2-perturbations of initial data. This argument seems to indicate that *a posteriori* error control in L_2 for approximate solutions of (7.1) would be impossible in the presence of shocks. The remarkable fact is, however, that such an error control in fact is possible to establish, if we use a proper stability concept and use the Galerkin orthogonality intrinsic in the finite element method, which we now proceed to demonstrate.

The *a posteriori* error estimate in L_2, for a finite element method for (7.1) to be presented, may be extended to the case of system of conservation laws in one space dimension. This yields a way out of the stalemate position of the classical approach to conservation laws, based on L_1-continuity, which is limited to the scalar case.

For simplicity, we shall consider a "semi-discrete" finite element method for (7.2) of the form of "exact transport + L_2-projection". cf. [J4]. The extension to the fully discrete case with discretization in space-time, using space-time elements, follows the same principles, see [JSz3].

Let $0 = t_0 < t_1 < t_2 <\ldots$ be a sequence of discrete time-levels and let, for each time interval $I_n = (t_{n-1}, t_n)$, a finite element space $V_n \subset H^1(\mathbb{R})$ be given, consisting of piecewise linear continuous functions on a finite element subdivision $\mathcal{T}_n = \{K\}$ of $\mathbb{R}$. For simplicity, we shall assume h to be independent of x and t, but this is not essential. We shall consider the following finite element method for (7.2): Find u_h such that, for $n = 1,2,\ldots$, $u_h|_{\mathbb{R}\times I_n}$ satisfies on $\mathbb{R} \times I_n$,

$$(7.3a)\quad \frac{\partial u_h}{\partial t} + \frac{\partial}{\partial x}\left[\frac{1}{2}u_h^2\right] - \hat{\epsilon}\,\frac{\partial^2 u_h}{\partial x^2} = 0\,,$$

$$(7.3b)\qquad u_h(x,t) \to 0,\quad x\to\pm\infty,\ t>0$$

$$(7.3c)\quad u_h^+(x,t_{n-1}) = P_n u_h^-(x,t_{n-1}),\ x\in\mathbb{R},$$

where $\hat{\epsilon} = Ch$,

$$u_h^{\pm}(x,t_{n-1}) = \lim_{s\to 0^+} u_h(x,t_{n-1}\pm s),$$

$u_n^-(x,0) = u_0(x)$, and $P_n: H^1(\mathbb{R}) \to V_n$ is the L_2-projection defined by

$$(7.4)\qquad \int_{\mathbb{R}} P_n v w\,dx = \int_{\mathbb{R}} v w\,dx \qquad \forall w \in V_n.$$

Note that (7.3) corresponds to a method of the form "exact transport + L_2-projection", similar to (for instance) the Godunov method, cf. [J4]. Note further that the viscosity coefficient $\hat{\epsilon}$ in (7.3) is chosen as $\hat{\epsilon} = Ch$, which makes (7.3) at most first order. Other more sophisticated choices of $\hat{\epsilon}$, with $\hat{\epsilon}$ depending on the residual of u_h (shock-capturing artificial viscosity) are possible, giving higher order methods, see above.

We shall now prove an *a posteriori* error estimate in $L_2(L_2)$ for the discretization error $e = u^{\hat{\epsilon}} - u_h$, where $u^{\hat{\epsilon}}$ satisfies (7.1) with $\hat{\epsilon} = \epsilon$, which appears to be or order $\mathcal{O}(h^2)$ in smooth parts and of order $\mathcal{O}(h^{\frac{1}{2}})$ in the presence of shocks. To obtain a complete estimate for $u^{\epsilon} - u_h$, we would also need to estimate $u^{\hat{\epsilon}} - u^{\epsilon}$, i.e., the effect of changing the viscosity coefficient in the continuous problem (7.1) from $\hat{\epsilon}$ to ϵ. We refer to [EJ7-8] and [JSz2] for details in this regard.

Let now $T = t_N > 0$ be a given final time, and let us seek an *a posteriori* error estimate for the quantity $\|e\|_{L_2(Q)}$, where $Q = \mathbb{R} \times (0,T)$. To this end, we introduce the following linearized dual problem, the stability properties of which are crucial:

(7.5a) $-\dfrac{\partial\phi}{\partial t} - a\dfrac{\partial\phi}{\partial x} - \hat{\epsilon}\dfrac{\partial^2\phi}{\partial x^2} = e$, in

(7.5b) $\phi(x,t)\to 0$, $x\to\pm\infty$, $0<t<T$,

(7.5c) $\phi(x,T)=0$, $x\in\mathbb{R}$,
Q,

where $a = (u^{\hat{\epsilon}} + u_h)/2$. Multiplying (7.5a) by e, integrating by parts over each $\mathbb{R}\times I_n$, $n = 1,\ldots, N$, and using that

$$a\frac{\partial\phi}{\partial x} = \left(\frac{1}{2}(u^{\hat{\epsilon}})^2 - \frac{1}{2}(u_h)^2\right)\frac{\partial\phi}{\partial x},$$

we obtain

$$\|e\|^2_{L_2(Q)}$$

$$= \int_Q e\left[-\frac{\partial\phi}{\partial t} - a\frac{\partial\phi}{\partial x} - \hat{\epsilon}\frac{\partial^2\phi}{\partial x^2}\right]dxdt$$

$$\sum_{n=1}^{N}\int_{\mathbb{R}}\int_{I_n}\left[\frac{\partial u^{\hat{\epsilon}}}{\partial t} + \frac{\partial}{\partial x}\frac{1}{2}(u^{\hat{\epsilon}})^2 - \hat{\epsilon}\frac{\partial^2 u^{\hat{\epsilon}}}{\partial x^2}\right.$$

$$\left. - \left[\frac{\partial u_h}{\partial t} + \frac{\partial}{\partial x}\frac{1}{2}u_h^2 - \hat{\epsilon}\frac{\partial^2 u_h}{\partial x^2}\right]\right]\phi\, dxdt$$

$$+ \sum_{n=1}^{N}\int_{\mathbb{R}}(I - P_n)u_h^-(x, t_{n-1})\phi(x, t_{n-1})dx,$$

so that, by (7.1a) with $\epsilon = \hat{\epsilon}$ and (7.3a), using the defining property (7.4) for P_n:

(7.6) $$\|e\|^2_{L_2(Q)} =$$

$$\sum_{n=1}^{N}\int_{\mathbb{R}}(I - P_n)u_h^-(x, t_{n-1})$$

$$(I - P_n)\phi(x, t_{n-1})dx.$$

To estimate the quantity $(I - P_n)\phi(\cdot,t_{n-1})$, we shall use the following strong stability estimate for the solution ϕ of the dual problem (7.5):

<u>Lemma 7.1.</u> Suppose $\dfrac{\partial a}{\partial x}$ is bounded from above. Then there is a constant C such that the solution ϕ of (7.5) satisfies

(7.7) $$\left\|\hat{\epsilon}\frac{\partial^2\phi}{\partial x^2}\right\|_{L_2(Q)}$$

$$+ \left\|\frac{\partial\phi}{\partial t} + a\frac{\partial\phi}{\partial x}\right\|_{L_2(Q)}$$

$$+ \sup_{0<t<T}\left\|\hat{\epsilon}^{\frac{1}{2}}\frac{\partial\phi}{\partial x}(\cdot,t)\right\|_{L_2(\mathbb{R})} \le C\|e\|_{L_2(Q)}.$$

<u>Proof.</u> Multiplying (7.5a) by $-\hat{\epsilon}\dfrac{\partial^2\phi}{\partial x^2}$, integrating over Q, and integrating by parts, we get

(7.8) $$-\int_0^T\frac{1}{2}\frac{d}{dt}\int_{\mathbb{R}}\hat{\epsilon}\left[\frac{\partial\phi}{\partial x}(\cdot,t)\right]^2 dxdt$$

$$+ \int_Q\left[\hat{\epsilon}\frac{\partial^2\phi}{\partial x^2}\right]^2 dxdt$$

$$+ \int_Q\frac{1}{2}\frac{\partial}{\partial x}\left[a\hat{\epsilon}\left[\frac{\partial\phi}{\partial x}\right]^2\right]dxdt$$

$$= \int_Q\frac{1}{2}\frac{\partial a}{\partial x}\hat{\epsilon}\left[\frac{\partial\phi}{\partial x}\right]^2 dxdt - \int_Q e\hat{\epsilon}\frac{\partial^2\phi}{\partial x^2}dxdt$$

which proves the lemma by a Grönvall inequality, since the third term on the left-hand side integrate to zero. ■

We now turn to the error representation (7.6). Using the standard estimates for the L_2-projection P_2,

$$\|(I - P_n)\phi(\cdot,t_{n-1})\|_{L_2}$$

$$\le Ch^2\left\|\frac{\partial^2\phi}{\partial x^2}(\cdot, t_{n-1})\right\|_{L_2},$$

we get by Cauchy's inequality

$$\|e\|_{L_2(\Omega)} \le \left[\sum_{n=1}^{N} \|(I - P_n)u_h^-(\cdot,t_{n-1})\|_{L_2}^2 k_n \frac{Ch^4}{\hat\epsilon^2 k_n^2}\right]^{\frac12} \left[\sum_{n=1}^{N} \left\|\hat\epsilon \frac{\partial^2\phi}{\partial x^2}(\cdot,t_{n-1})\right\|_{L_2}^2 k_n\right]^{\frac12},$$

where $k_n = t_n - t_{n-1}$. If we now assume that $k_n \ge Ch$ for some positive constant C, and recall that $\hat\epsilon = Ch$, we have that

$$\|e\|_{L_2(Q)}^2$$

$$\le CA\left[\sum_{n=1}^{N} \|(I-P_n)u_h^-(\cdot,t_{n-1})\|_{L_2}^2 k_n\right]^{1/2},$$

where

$$A = \left[\sum_{n=1}^{N} \left\|\hat\epsilon \frac{\partial^2\phi}{\partial x^2}(\cdot,t_{n-1})\right\|_{L_2}^2 k_n\right]^{1/2}$$

may be viewed as an approximation of $\|\hat\epsilon \frac{\partial^2\phi}{\partial x^2}\|_{L_2(Q)}$, cf. below. Using Lemma 4.1, we thus conclude that

(7.9) $\quad \|e\|_{L_2(Q)} \le C\|(I - P)u_h^-\|_{L_2(Q)}$,

where, for convenience of notation, we have defined for $t \in I_n$,

$$(I - P)u_h^-(x,t) = (I - P_n)u_h^-(x, t_{n-1})$$

so that

$$\|(I - P)u_h^-\|_{L_2(Q)}$$

$$= \left[\sum_{n=1}^{N} \|(I - P_n)u_h^-(\cdot, t_{n-1})\|^2 k_n\right]^2.$$

We have thus arrived at the surprisingly simple *a posteriori* error estimate (7.9), which can be made more concrete by estimating the projection error $(I - P_n)u_h^-$ as follows ([JSz2]):

$$\|(I - P_n)u_h^-(\cdot,t_{n-1})\|_{L_2}$$

$$\le Ch^2\|D_h^2 u_h^-(\cdot,t_{n-1})\|_{L_2},$$

where, with $K = (x_1,x_2) \in T_{n-1}$,

$$D_h^2 u_h^-(x,t_{n-1})|_K$$

$$= h^{-1} \max_{i=1,2} \left|\left[\frac{\partial u_h^-}{\partial x}(x_i,t_{n-1})\right]\right|$$

$$+ \max_K \left|\frac{\partial^2 u_h^-}{\partial x^2}(\cdot,t_{n-1})\right|$$

with $[\frac{\partial u_h^-}{\partial x}(x_i,t_{n-1})]$ the jump in the derivative $\frac{\partial u_h^-}{\partial x}$ at (x_i, t_{n-1}). Extending $D^2 u_h^-$ to $(0,T)$ by defining

$$D_h^2 u_h^-(x,t) = D^2 u_h^-(x,t_{n-1}) \text{ for } t \in I_n,$$

we can thus express the *a posteriori* error estimate (7.9) alternatively as follows:

(7.10) $\quad \|e\|_{L_2(Q)} \le C\|h^2 D_h^2 u_h^-\|_{L_2(Q)}$.

Some comments are in order. First, the assumption used above, that $A \approx \|\hat\epsilon \frac{\partial^2\phi}{\partial x^2}\|_{L_2(Q)}$ can easily be avoided, using also the presence of the other terms in the strong stability estimate (7.7), see [JSz2] for details. Secondly, the *a posteriori* estimate (7.10) appears to be optimal, i.e., in particular second order accurate for smooth solutions. This is a consequence of the choice of $\hat\epsilon$ as $\hat\epsilon = Ch$, which itself would correspond to an $\mathcal{O}(h)$ perturbation for smooth solutions, and thus reduce the accuracy of the total approximation to first order. With the more elaborate choice of $\hat{\epsilon}$ as, for instance, $\hat{\epsilon} = C \max(h^2 R(u_h), h^{3/2})$, where $R(u_h)$ is the residual of u_h defined below, which is a typical choice of the artificial viscosity in the SD-method (see above), we would obtain a method of total accuracy $\mathcal{O}(h^{3/2})$ for smooth solutions satisfying the following *a posteriori* error estimate (cf. (1.5))

$$(7.11) \quad \|e\|_{L_2(Q)} \le C\|\min(1, h^{1/2}R(u_h))\|_{L_2(Q)}$$

or

$$(7.12) \quad \|e\|_{L_2(Q)} \le C\|\min(1, h^{3/2}D^2 u_h^-)\|_{L_2(Q)},$$

where we define

$$R(u_h) = (I - P_n)u_h^-(\cdot, t_{n-1})/h \quad \text{on } I_n.$$

We summarize the results obtained for Burgers' equation as follows:

<u>Theorem 7.2.</u> Let u_h be the solution of (4.3) and u^ϵ that of (7.1) with $\epsilon = \hat{\epsilon} = Ch$ and $k_n \ge Ch$. Suppose that $\frac{\partial a}{\partial x} = \frac{1}{2}\frac{\partial}{\partial x}(u^{\hat{\epsilon}} + u_h)$ is bounded from above, and that $u^{\hat{\epsilon}}$ and u_h are bounded. Then there are constants C such that

$$(7.13) \quad \|u^{\hat{\epsilon}} - u_h\|_{L_2(Q)} \le C\|(I - P)u_h^-\|_{L_2(Q)} \le C\|h^2 D^2 u_h^-\|_{L_2(Q)}.$$

<u>Remark 7.1.</u> Note that the estimates (7.13) indicate $\mathcal{O}(\sqrt{h})$ accuracy (which is optimal) in the presence of shocks since, in an $\mathcal{O}(h)$-neighbourhood of the shock, the integrands on the right-hand side would be of order $\mathcal{O}(1)$.

Remark 7.2. The dual problem (7.5) may be L_2-unstable, since multiplication of (7.5a) with ϕ gives

$$-\frac{1}{2}\frac{d}{dt}\int_{\mathbb{R}} \phi^2(x,t)dx + \hat{\epsilon}\int_{\mathbb{R}} \left[\frac{\partial \phi}{\partial x}(x,t)\right]^2 dx$$

$$= -\frac{1}{2}\int_{\mathbb{R}} \frac{\partial a}{\partial x}\phi^2(x,t)dx + \int_{\mathbb{R}} e(x,t)\phi(x,t)dx,$$

which may lead to instability if $\frac{\partial a}{\partial x}$ is large negative, cf. the discussion above for the linearized problem (7.2).

For analogs of (7.13) for full discretizations of (7.1) and extensions to systems of conservation laws in one space dimension, we refer to [JSz3], where we also consider the case of rarefaction waves with $\frac{\partial a}{\partial x}$ possibly large for t small, $(\frac{\partial a}{\partial x} \le 1/t)$, using a weighted norm technique. The *a posteriori* error estimate of Theorem 7.2 may be extended to systems of conservation laws in one dimension under appropriate assumptions including the presence of shocks, and rarefaction waves. As far as we know, these results are the first to show that *a posteriori* error control for systems of conservation laws is possible. To establish the crucial stability estimates in the system case corresponding to Lemma 7.1, diagonalization together with a weighted norm technique is used.

The techniques for proving *a posteriori* error estimates for conservation laws indicated above may formally be extended to systems of conservation laws in several dimensions, leading to *a posteriori* error estimates of e.g., the form (0.10), if the corresponding linearized dual problem satisfies strong stability estimates analogous to (7.7). In [HJ] we give computational results for the corresponding adaptive algorithms in the case of time-dependent compressible flow in two dimensions. Below we present a corresponding result for a stationary shock reflection problem. The question if the linearized dual problem satisfies the strong stability estimates in the case of systems of conservation laws (or the incompressible Navier-Stokes equations)in several dimensions is theoretically very complex, but, as indicated, may probably be tested computationally. We plan to give more details on this topic in future work.

8. NUMERICAL RESULTS

In this section we present some numerical results for SD and CSD-methods applied to

(8.1) stationary convection-diffusion

(8.2) non-stationary convection-diffusion

(8.3) the incompressible Navier-Stokes equations with free boundary (fountain flow)

(8.4) the compressible Euler and Navier-Stokes equations

9. CONCLUSION. PROSPECTS FOR THE FUTURE

We have given an overview of the SD-method as a general unified approach to CFD based on the principles: modified Galerkin + space-time finite elements.

The SD-method may be viewed to generalize all the main classical techniques of CFD such as finite difference, finite volume, particle and shock-fitting methods. The SD-method proposes a solution to the fundamental problems of design of artificial viscosity, combination of Eulerian and Lagrangean approaches, adaptive quantitative error control, and of course is applicable on general unstructured meshes.

The method appears to have a strong potential, in particular in adaptive form. Various features of the SD-method are today used in several commercial and research codes. A rapid development into full exploitation of SD-features such as space-time elements and adaptivity in these codes is to be expected.

REFERENCES

[EJ1] K. Eriksson and C. Johnson, Error estimates and automatic time step control for nonlinear parabolic problems, I, SIAM J. Numer. Anal. 24 (1987), 12-23.

[EJ2] K. Eriksson and C. Johnson, An adaptive finite element method for linear elliptic problems, Math. Comp. 50 (1988), 361-383.

[EJ3] K. Eriksson and C. Johnson, Adaptive finite element methods for parabolic problems I: A linear model problem, SIAM J. Numer. Anal. 28 (1991), 43-77.

[EJ4] K. Eriksson and C. Johnson, Adaptive finite element methods for parabolic problems II: A priori error estimates in $L_\infty(L_2)$, Technical report, Chalmers University of Technology, 1992.

[EJ5] K. Eriksson and C. Johnson, Adaptive finite element methods for parabolic problems III: Time steps variable in space and non-coercive problems, Technical report, Chalmers University of Technology, 1992.

[EJ6] K. Eriksson and C. Johnson, Adaptive finite element methods for parabolic problems IV: Non-linear problems, Technical report, Chalmers University of Technology 1992.

[EJ7] K. Eriksson and C. Johnson, Adaptive streamline diffusion finite element methods for stationary convection-diffusion problems, Math. Comp. (to appear).

[EJ8] K. Eriksson and C. Johnson, Adaptive streamline diffusion finite element methods for time-dependent convection-diffusion problems, Math. Comp. (to appear).

[Ha1] P. Hansbo, Adaptivity and Streamline Diffusion Procedures in the Finite Element Method, Thesis, Publication 89:2, Dept. of Structural Mechanics, Chalmers Univ. of Technology.

[Ha2] P. Hansbo, The characteristic streamline diffusion method for convection-diffusion problems, Preprint 1990-25, Mathematics Dept., Chalmers Univ. of Technology.

[Ha3] P. Hansbo, The characteristic streamline diffusion method for the time-dependent incompressible Navier-Stokes equations, Preprint 1991-14, Mathematics Dept., Chalmers Univ. of Technology.

[Ha4] P. Hansbo, Explicit streamline diffusion finite element methods for the compressible Euler equations in conservation variables, Preprint 1991-3, Mathematics Dept. Chalmers Univ. of Technology.

[HJ] P. Hansbo and C. Johnson, Adaptive streamline diffusion methods for compressible flow using conservation variables, Comput. Methods Appl. Mech. Engrg. 87 (1991), 267-280.

[HJS] P. Hansbo, C. Johnson and A. Szepessy, Shock-capturing streamline diffusion finite element methods for nonlinear conservation laws, in: Proceedings of the 7th International Conference on Finite Element Methods in Flow problems, UAH Press, Huntsville (1989), 377-382.

[HS] P. Hansbo and A. Szepessy, A velocity pressure streamline diffusion finite element method for the incompressible Navier-Stokes equations, to appear in Comp. Meth. in Appl. Mech. Engrg. 84(1990), 175-192.

[Hu] Hulbert, G. and Hughes, T. Space-time finite element methods for second order hyperbolic equations, Comp. Meth. Appl. Mech. Engrg.

[H1] T.J.R. Hughes, L.P. Franca and M. Mallet, A new finite element formulation for computational fluid dynamics: I Symmetric forms of the compressible Euler and Navier-Stokes equations and the second law of thermodynamics, Computer methods in applied mechanics and engineering 54 (1986) 223-234.

[H2] T.J.R. Hughes, M. Mallet and A. Mizukami, A New Finite element formulation for Computational Fluid Dynamics: II. Beyond SUPG, Comput. Meths. Appl. Mech. Engrg. Vol. 54 (1986), 341-355.

[H3] T.J.R. Hughes, M. Mallet, A New Finite Element Formulation for Computational Fluid Dynamics: III The General Streamline Operator for Multidimensional Advective-Diffusive Systems, Comput. Meths. Appl. Mech. Engrg. Vol 58 (1986), 329-336.

[H4] T.J.R. Hughes and M. Mallet, A New Finite Element Formulation for Computational Fluid Dynamics: IV. A Discontinuity-Capturing Operator for Multidimensional Advective-Diffusive Systems, Comput. Meth. Appl. Mech. Engrg., Vol 58 (1986), 329-336.

[H5] T.J.R. Hughes, L.P. Franca and M. Balestra, A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuska-Brezzi Condition: A Stable Petrov-Galerkin Formulation of the Stokes-Problem Accomodating Equal-order Interpolations, Comput. Meths. Appl. Mech. Engrg., Vol. 59 (1986), 85-99.

[HT] T.J.R. Hughes and T.E. Tezduyar, Finite element methods for first-order nyperbolic systems with particular emphasis on the compressible Euler equations, Comput. Methods Appl. Mech. Engrg. 45 (1984), 217-284.

[HB1] T.J.R. Hughes and A. Brooks, A multidimensional upwind scheme with no crosswind diffusion, AMD- vol 34, Finite Element Methods for Convection Dominated Flows, e.d. T.J.R. Hughes, ASME New York 1979.

[HB2] T.J.R. Hughes, A. Brooks, Streamline Upwind;Petrov-Galerkin Formulations for Convection Dominated Flows with Particular Emphasis on the Incompressible Navier-Stokes Equations,

[J1] C. Johnson, Error estimates and adaptive time step control for a class of one step methods for stiff ordinary differential equtions, SIAM J. Numer. Anal. 25 (1988), 908-926.

[J2] C. Johnson, Adaptive finite element methods for diffusion and convection problems, Comput. Methods Appl. Mech. Engrg. 82 (1990), 301-322.

[J3] C. Johnson, The characteristic streamline diffusion finite element method, Matematica Aplicada e Computacional, 10 (1992), Proc. from Conf. on Innovative Finite Element Methods, Rio de Janeiro, Dec 1989.

[J4] C. Johnson, A new approach to algorithms for convection problems which are based on exact transport + projection, Comput. Methods Appl. Mech. Engrg. (to appear).

[J5] C. Johnson, Discontinuous Galerkin finite element methods for second order hyperbolic problems, Technical report, Chalmers University of Technology, Göteborg, 1991.

[J6] C. Johnson, The streamline diffusion method for compressible and incompressible flow, Proc. Finite elements in Fluids VII, Huntsville 1989, Hemisphere 1992.

[JH] C. Johnson and P. Hansbo, Adaptive finite element methods in computational mechanics, to appear in Comp. Meth. Appl. Mech. Engrg, Proc. from Workshop on Reliability in Computational Mechanics, Cracow, Oct. 1991.

[JJ] C. Johnson and J. Jaffre, The Discontinuous Galerkin method for conservation laws, to appear.

[JN] C. Johnson and U. Nävert, An analysis of some finite element methods for advection-diffusion problems, in: O. Axelsson, L.S. Frank and A. van der Sluis, eds., Analytical and Numerical Approaches to Asymptotic Problems in Analysis (North-Holland, Amsterdam, 1981).

[JNP] C. Johnson and U. Nävert and J. Pitkäranta, Finite element methods for linear hyperbolic problems, Comput. Methods Appl. Mech. Engrg. 45 (1984) 285-312.

[JS] C. Johnson and J. Saranen, Streamline diffusion methods for the incompressible Euler and Navier-Stokes equations, Math. Comp. 47 (1986) 1-18.

[JSz1] C. Johnson and A. Szepessy, On the convergence of a finite element method for a nonlinear hyperbolic conservation law, Math. Comp. 49 (1987) 427-444.

[JSz2] C. Johnson and A. Szepessy, On the convergence of streamline diffusion finite element methods for hyperbolic conservation laws, in: T.E. Tezduyar and T.J.R. Hughes, eds., Numerical Methods for Compressible Flow - Finite Difference, Finite Element and Volume Techniques, AMD-Vol. 78 (ASME, New York, 1986).

[JSz3] C. Johnson and A. Szepessy, Adaptive streamline diffusion methods for conservation laws (to appear).

[JSW] C. Johnson, A.H. Schatz and L.B. Wahlbin, Crosswind smear and pointwise errors in streamline diffusion finite element methods, Math. Comp. 47 (1987) 25-38.

[JSzH] C. Johnson, A. Szepessy and P. Hansbo, On the convergence of shock-capturing streamline diffusion finite element methods for hyperbolic conservation laws, Math. Comp. 54 (1990) 107-129.

[N] U. Nävert, A finite element method for convection-diffusion problems. Thesis, Department of Computer Science, Chalmers University of Technology, 1982.

[Sh] F. Shakib, Finite element analysis of the compressible Euler and Navier-Stokes equations, Thesis, Stanford University, Stanford, CA, 1988.

[ShHJ] F. Shakib, T.J.R. Hughes and Z. Johan, A new finite element method for computational fluid dynamics: X. The compressible Euler and Navier-Stokes equations, Comput. Methods Appl. Mech. Engrg. 89 (1991), 141-219.

[Sz1] A. Szepessy, Convergence of the streamline diffusion finite element method for conservation laws, Thesis, Department of Mathematics, Chalmers University of Technology, 1989.

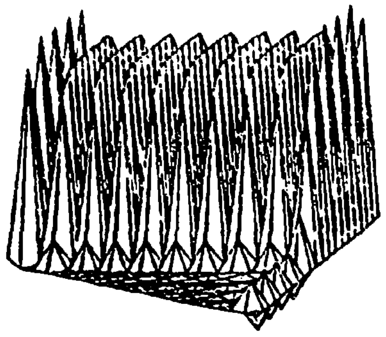
[Sz2] A. Szepessy, Convergence of a shock-capturing streamline diffusion finite element method for a scalar conservation law in two space dimensions, Math. Comp. 53 (1989) 27-545.

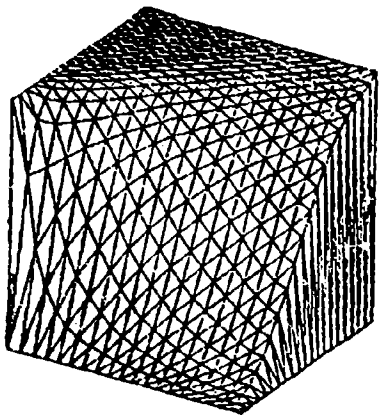
[Sz3] A. Szepessy, Nonlinear stability of shock waves of a finite element method for systems of conservation laws, to appear in Comm. Pure Appl. Math.

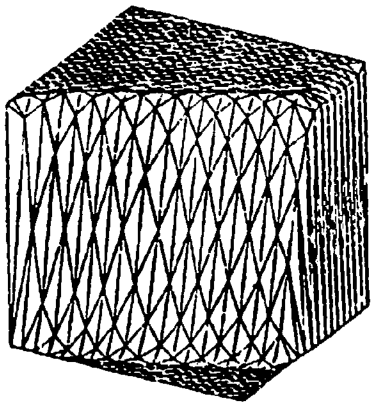
[Sz4] A. Szepessy, Convergence of a streamline diffusion finite element method for scalar conservation laws with boundary conditions, Mathematical Modelling and Numerical Analysis 25 (1991), 749-782.

[SzX] A. Szepessy and Z. Xin, Nonlinear stability of viscous shock waves, Trifa-NA-9201, Royal Inst. of Technology, Stockholm.

[TLB] T. Tezduyar, J. Lion and M. Behr, A new strategy for finite element computations involving moving boundaries and interfaces - The DSD/ST procedure: I. The concept and preliminary numerical tests. II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders. To appear in Comp. Meth. Appl. Mech. Engrg.


Standard Galerkin

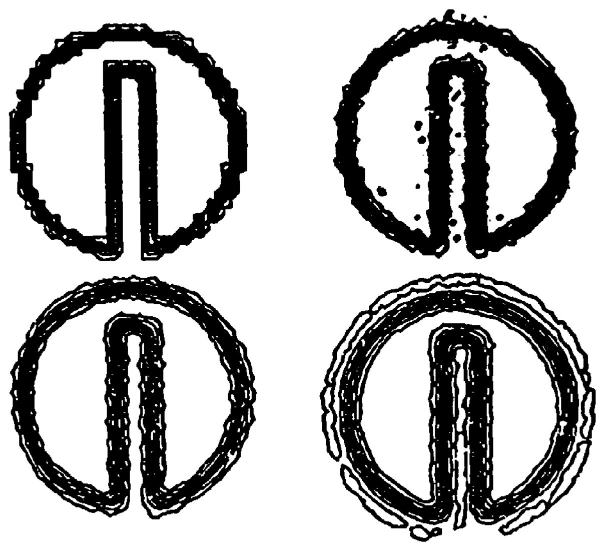

Classical artificial diffusion
Upwind method


The sd-method

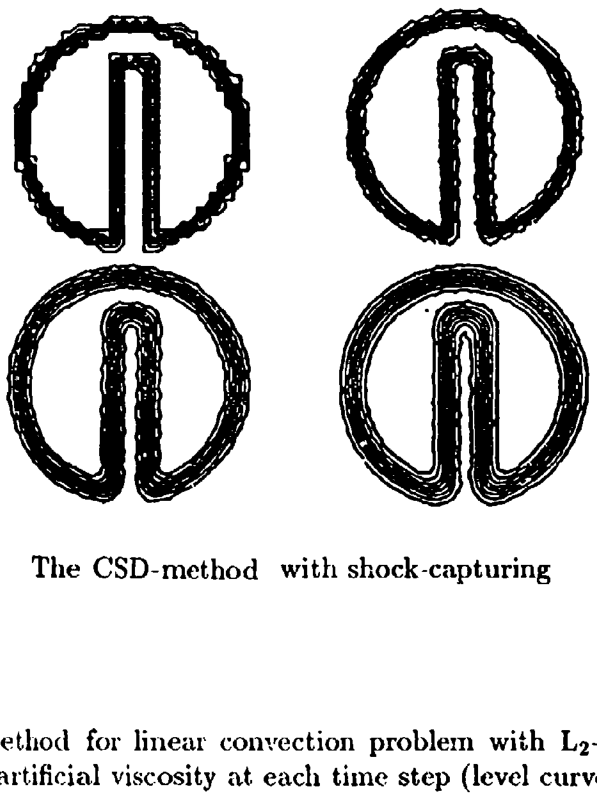
$\beta \cdot \nabla - \epsilon \Delta u = 0 \qquad \text{in } (0,1)^2,$
u = 1 if x = 0 or y = 1
u = 0 if y = 0 or x = 1

$\beta = (2, 1)$

FIG. 1 - Comparison of Standard Galerkin, Classical artificial diffusion and the SD-method for a stationary convection-diffusion problem with internal layer and boundary layer

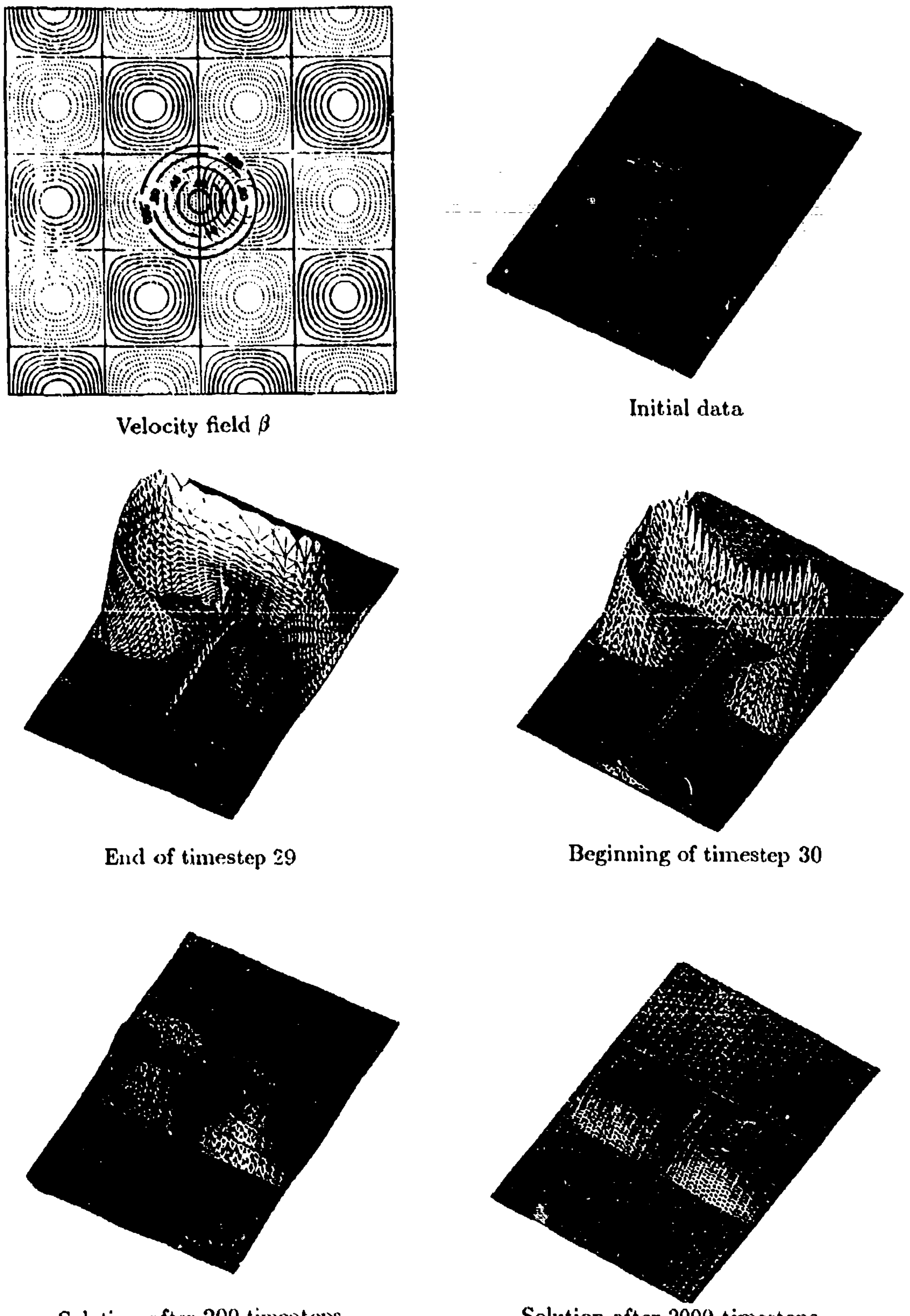


The CSD-method without shock-capturing



The CSD-method with shock-capturing

FIG. 2 - The CSD-method for linear convection problem with L_2-projection with and with shock-capturing artificial viscosity at each time step (level curves of initial data and CSD-solution after 5, 50 and 100 time steps)



Velocity field β

Initial data

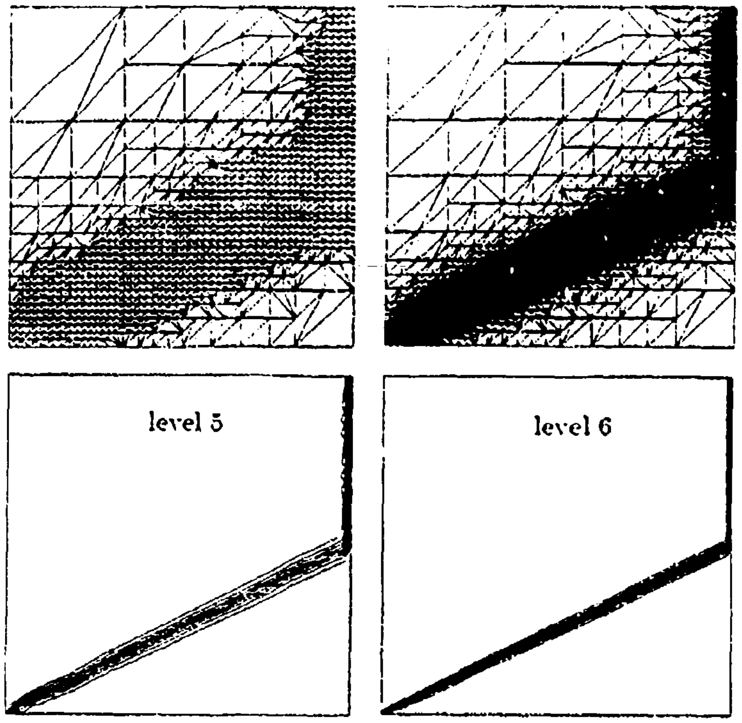
End of timestep 29

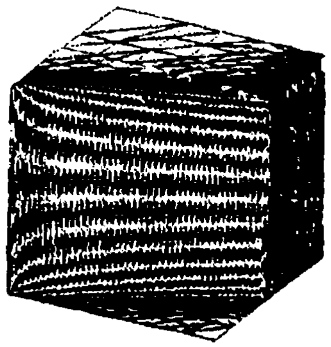
Beginning of timestep 30

Solution after 200 timesteps

Solution after 2000 timesteps

FIG. 3 - The CSD-method for convection of initial conical distribution in complex velocity field β consisting of rotating and counterrotating vortices

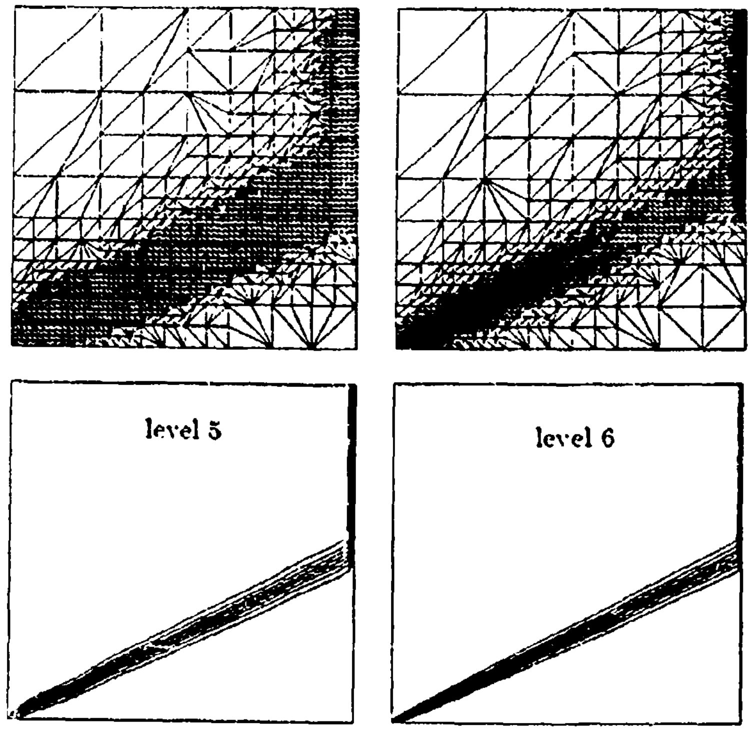


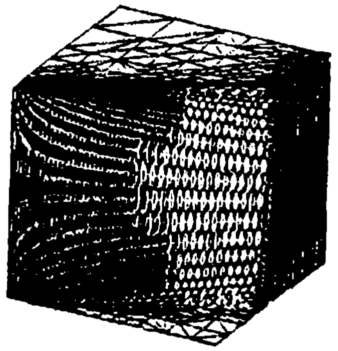


$\beta \cdot \nabla u - \epsilon \Delta u = 0 \quad \text{in } (0,1)^2$
$u = 1$ if $x = 0$ or $y = 1$
$u = 0$ if $y = 0$ or $x = 1$
$\beta = (2,1)$

LEVEL	NODES	ELEMENTS	L2-NORM ERROR	APPROX. L2-ERR
1	18	18	0.291093E+00	0.295545E+00
2	48	60	0.213298E+00	0.232075E+00
3	142	242	0.163790E+00	0.208038E+00
4	382	784	0.121565E+00	0.163415E+00
5	1027	1356	0.985245E-01	0.121770E+00
6	2419	4789	0.677035E-01	0.102815E+00

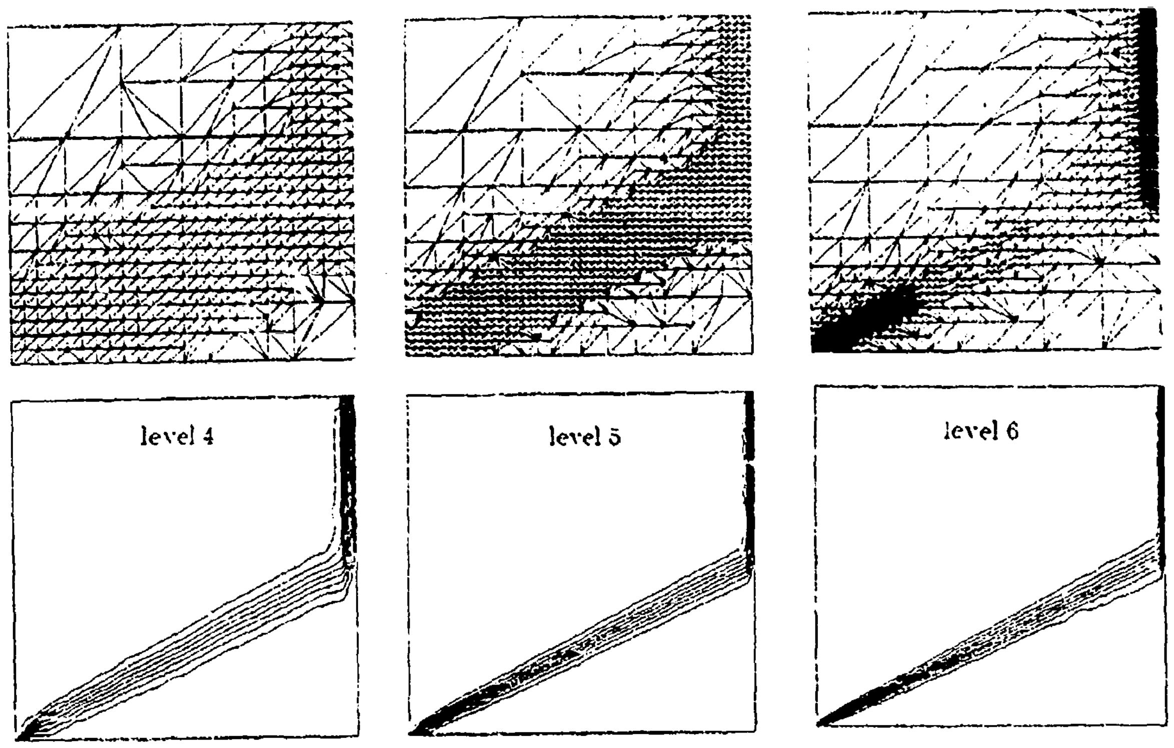
FIG. 4a - Stationary convection-diffusion problem with internal and boundary layer corresponding to $\epsilon = 10^{-6}$. Adaptivity based on (1.23) with TOL = 0.1

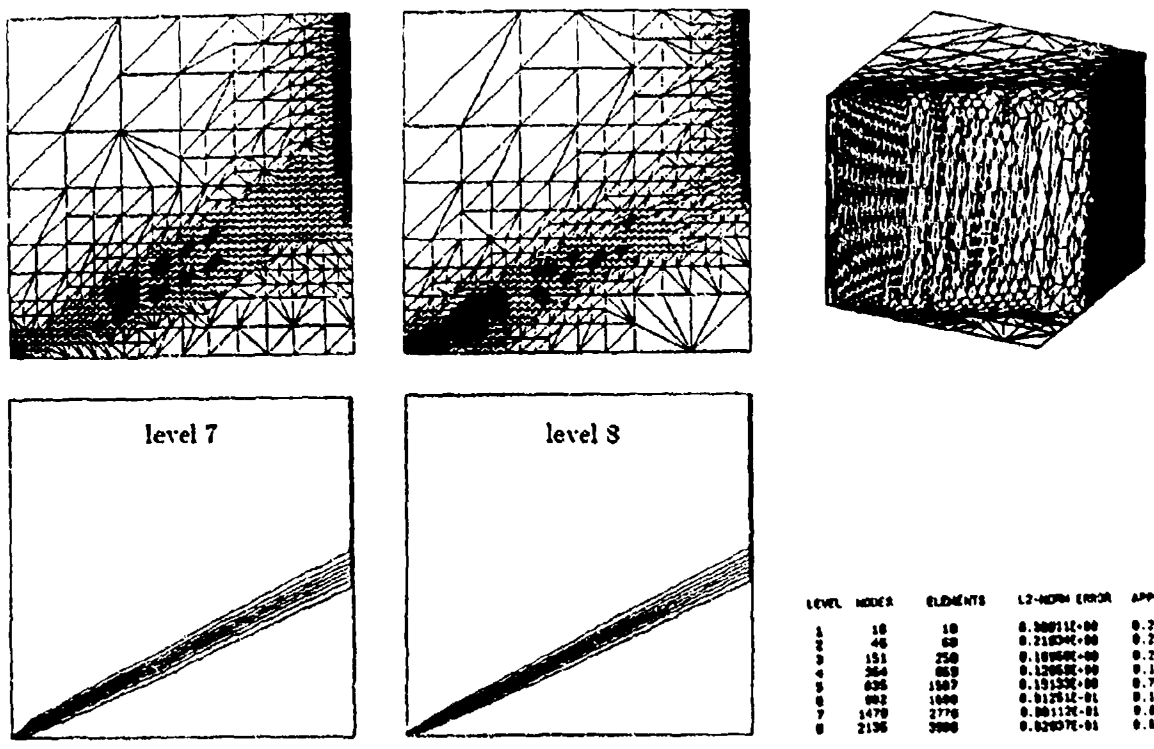




LEVEL	NODES	ELEMENTS	L2-NORM ERROR	APPROX. L2-ERR
1	26	32	0.256108E+00	0.264958E+00
2	78	124	0.200651E+00	0.229327E+00
3	235	417	0.150108E+00	0.180330E+00
4	839	1203	0.115158E+00	0.111482E+00
5	1303	2163	0.933272E-01	0.879500E-01
6	1873	3982	0.827046E-01	0.453956E-01

Fig. 4b - As in Fig. 4a with now $\epsilon = 10^{-4}$ and TOL = 0.05





LEVEL	NODES	ELEMENTS	L2-NORM ERROR	APPROX. L2-ERR.
1	10	10	0.300[illegible]E+00	0.295311E+00
2	46	60	0.218[illegible]E+00	0.23643E+00
3	151	250	0.1[illegible]E+00	0.20240E+00
4	364	659	0.1206[illegible]E+00	0.14990E+00
5	636	1307	0.1513[illegible]E+00	0.73539E-01
6	902	1600	0.91251E-01	0.10334E+00
7	1470	2776	0.80112E-01	0.62981E-01
8	2136	3900	0.82037E-01	0.64550E-01

FIG. 4c - As in Fig. 4a with now $\epsilon = 10^{-3}$ and with the extra requirement $\hat{\epsilon} = \epsilon$, and TOL = 0.15

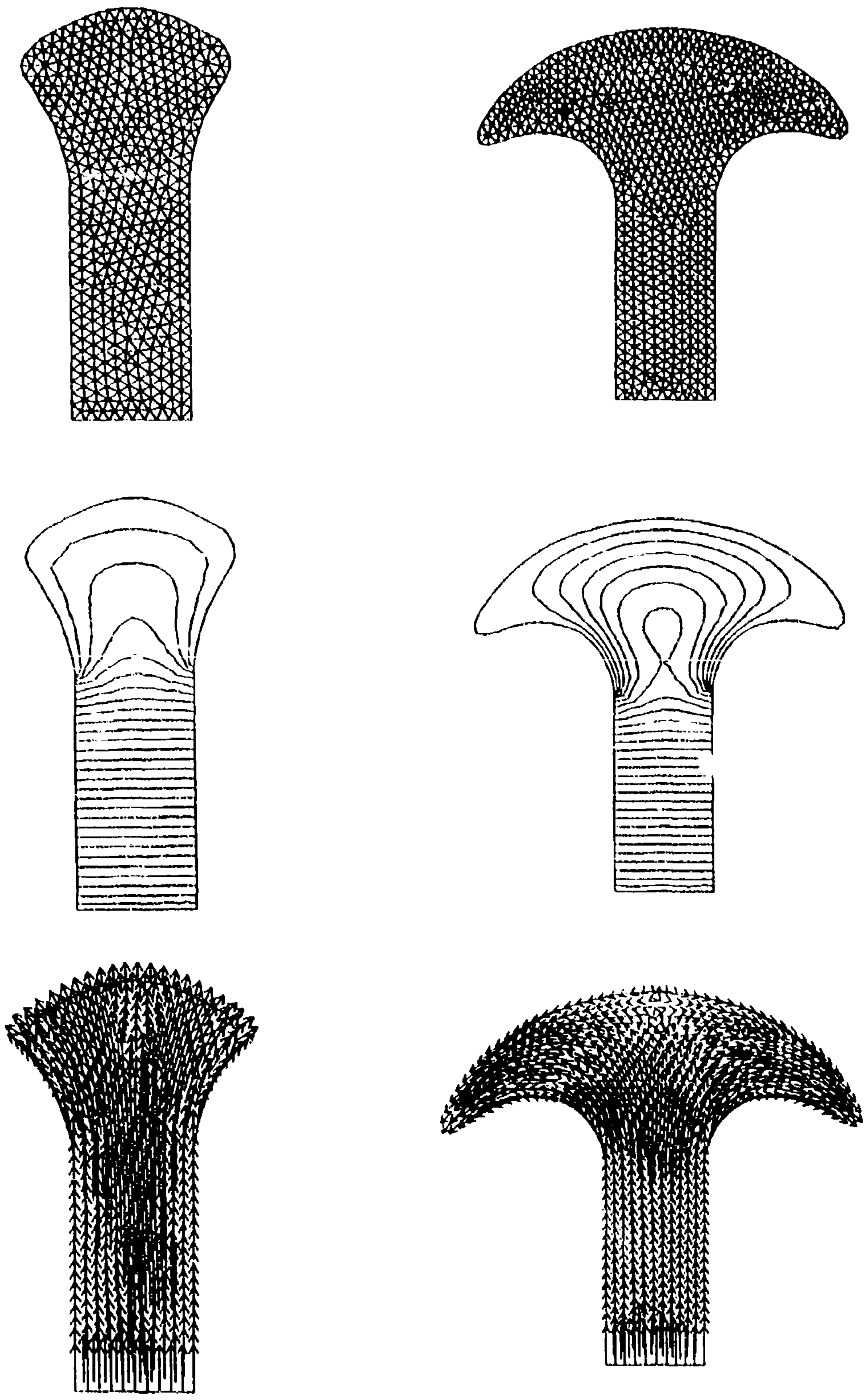


FIG. 5 - The CSD-method for the incompressible Navier-Stokes equations with free boundary (fountain flow)

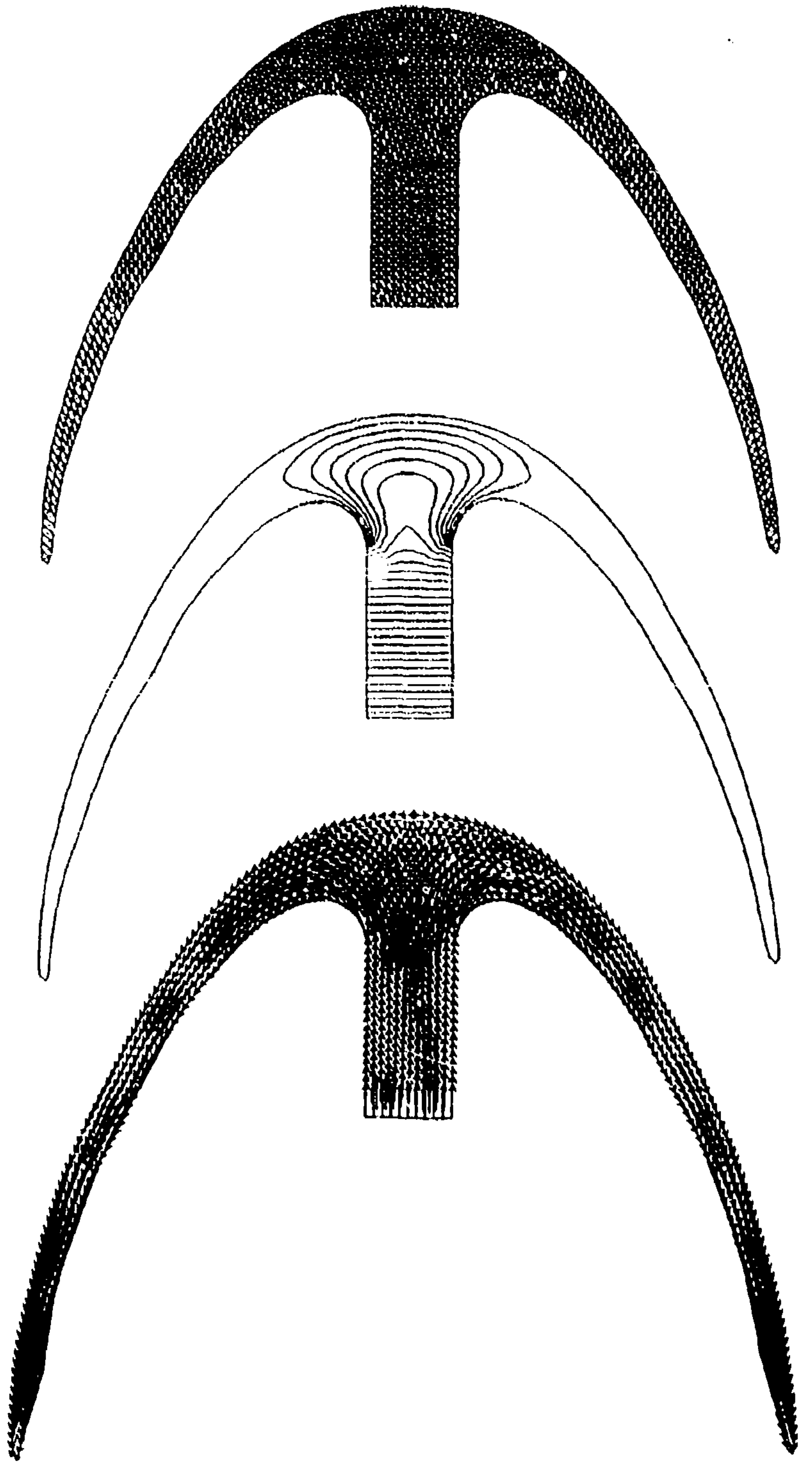


FIG. 5 cont'd

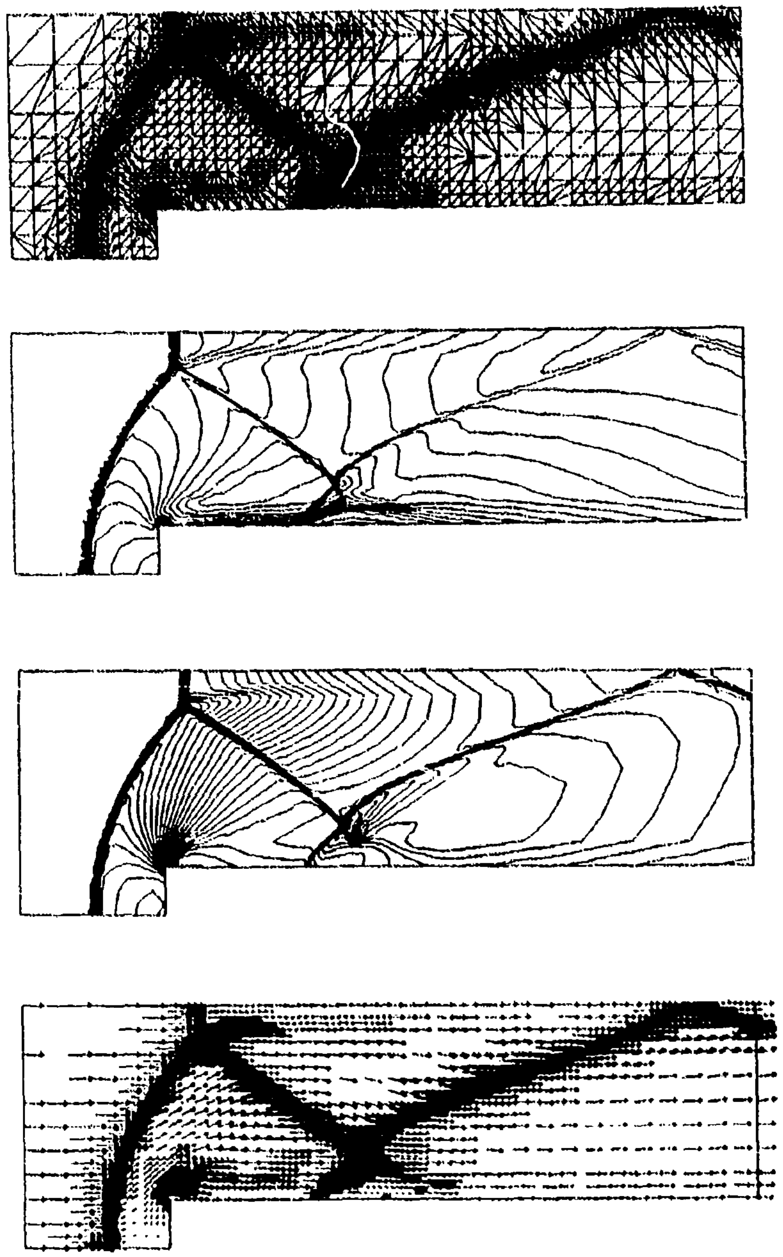
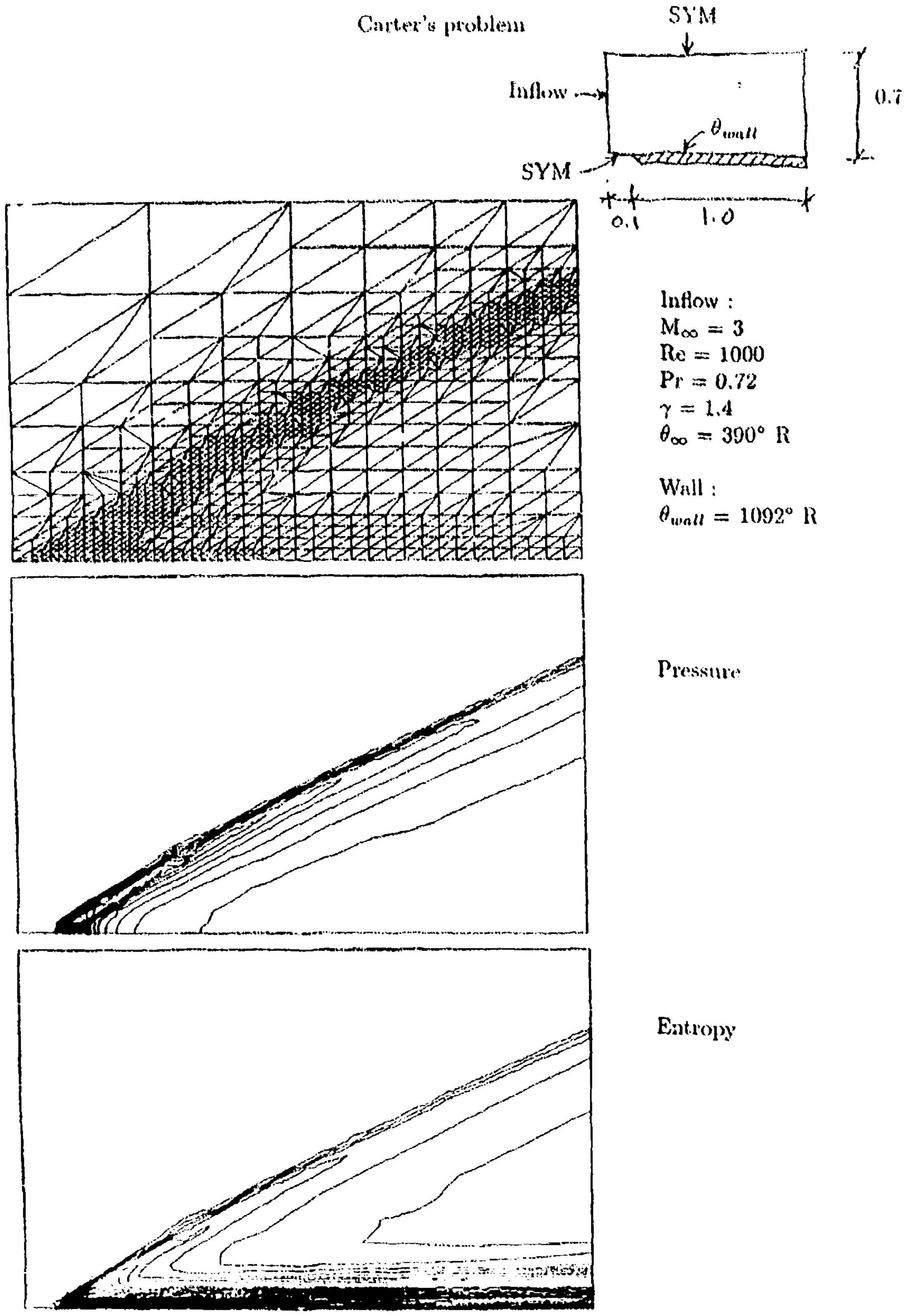


FIG. 6 - Adaptive SD-method for the compressible Euler equations (fully explicit version (5.7)).

Mach 3 channel flow with a step up.
Adaptivity according to (0.10)



Inflow :
$M_\infty = 3$
$Re = 1000$
$Pr = 0.72$
$\gamma = 1.4$
$\theta_\infty = 390°$ R

Wall :
$\theta_{wall} = 1092°$ R

FIG. 7 - Adaptive SD-method for the compressible Navier-Stokes equations. Flow over a flat plate

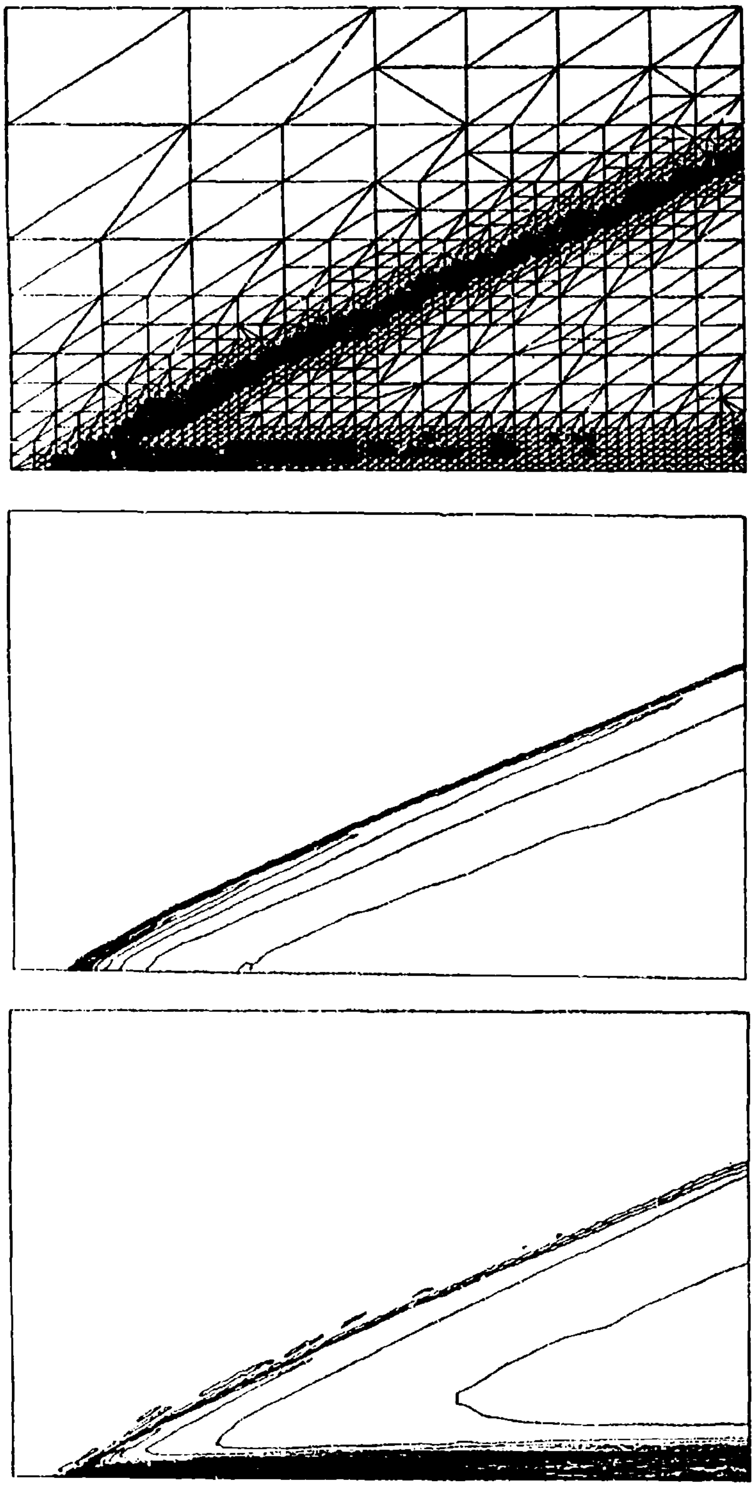
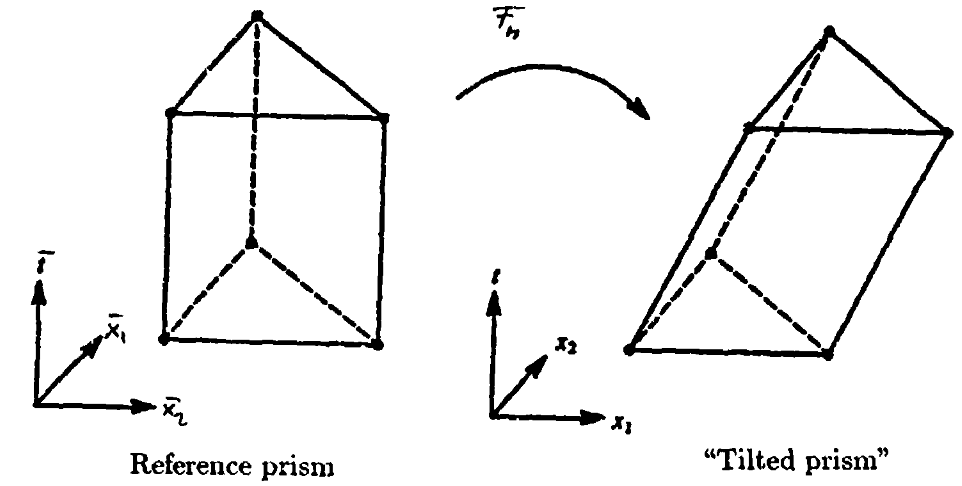
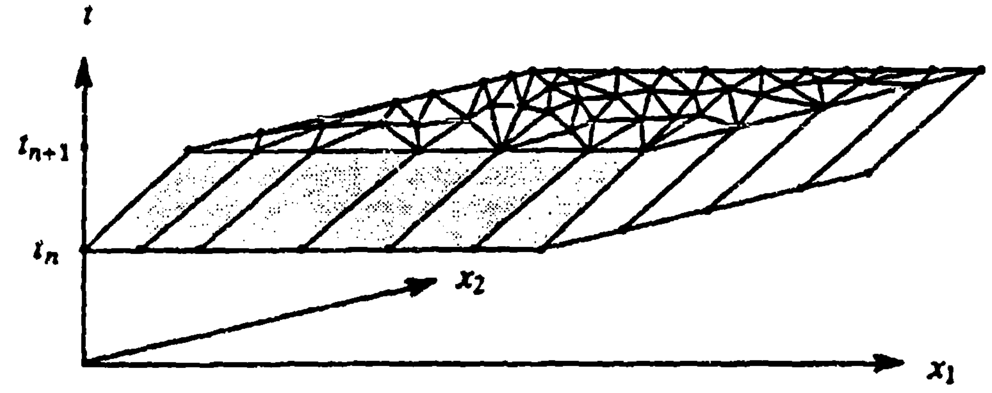


FIG. 7 cont'd





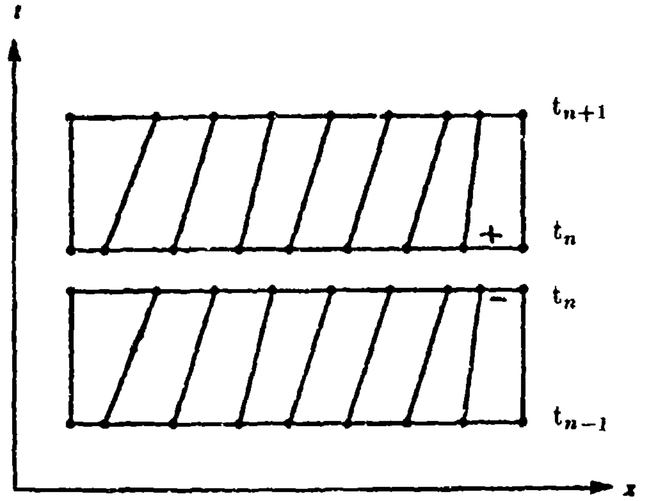


FIG. 8 - Space-time discretization of the SD-method

FINITE ELEMENT METHODS FOR FLUIDS

by

Thomas J.R. Hughes
Division of Applied Mechanics
Stanford University
Stanford, CA 94305-4040
United States

Outline

- Stabilized methods.
- Space-time formulations.
- Symmetric linear advective-diffusive *systems.*
- Incompressible Euler and Navier-Stokes equations; Stokes Problem.
- Compressible Euler and Navier-Stokes equations; Entropy variables.
- Nonlinear operators and shock-capturing.
- Solution algorithms.
- Examples.

Stabilized methods

Example: Scalar, steady advection-diffusion equation.

$$\mathcal{L}u \overset{\text{def}}{=} \boldsymbol{a}\cdot\nabla u - \nabla\cdot\kappa\nabla u = f \qquad \text{on } \Omega \subset \mathbf{R}^d$$

Assume $\nabla\cdot\boldsymbol{a} = 0$, $\kappa > 0$.

Consider $u = 0$ on Γ (Dirichlet problem).

Remark:

$\alpha = h|\boldsymbol{a}|/(2\kappa)$ = element Peclet number

h = element mesh parameter

Interested in $0 < \alpha < \infty$; $\alpha \gg 1$ is viewed as "hard."

$\mathcal{V}^h$ = typical finite element space of continuous, piecewise polynomials of order k.

Point of departure: **Galerkin's method**

Find $u^h \in \mathcal{V}^h$ such that for all $w^h \in \mathcal{V}^h$,

$$B(w^h, u^h) = L(w^h)$$

where

$$B(w^h, u^h) \overset{\text{def}}{=} \int_\Omega (-\boldsymbol{a}\cdot\nabla w^h\, u^h + \nabla w^h \cdot \kappa\nabla u^h)\, d\Omega$$

$$L(w^h) \overset{\text{def}}{=} \int_\Omega w^h f\, d\Omega$$

Remark: Galerkin's method possesses poor stability properties for $\alpha > 1$. Spurious oscillations are generated by unresolved internal and boundary layers.

Galerkin/least-squares

$$B(w^h, u^h) + \sum_e \int_{\Omega^e} \mathcal{L}w^h\, \tau(\mathcal{L}u^h - f)\, d\Omega = L(w^h)$$

SUPG

$$B(w^h, u^h) + \sum_e \int_{\Omega^e} \boldsymbol{a}\cdot\nabla w^h\, \tau(\mathcal{L}u^h - f)\, d\Omega = L(w^h)$$

Classical artificial diffusion

$$B(w^h, u^h) + \int_\Omega \nabla w^h \cdot \kappa^h \nabla u^h\, d\Omega = L(w^h)$$

Notations:

Ω^e = domain of eth element.

κ^h = artificial diffusivity, typically $O(h)$.

τ = parameter determined by convergence analysis.

$= O(h/|\boldsymbol{a}|)$ for α large.

$= O(h^2/\kappa)$ for α small.

Remarks:

1. The additional terms improve upon the stability of Galerkin's method.
2. The classical artificial diffusion method amounts to overkill. Accuracy is limited to first-order, independent of k.
3. Galerkin/least-squares and SUPG are satisfied if $u^h \leftarrow u$ ("residual methods"); unlike classical artificial diffusion.
4. Good stability and higher-order accuracy are combined in Galerkin/least-squares and SUPG.
   - Global error estimates for *smooth* solutions: $k + \frac{1}{2}$ (at least) in $L_2(\Omega)$, usually $k + 1$ in practice.
   - For *rough* solutions, same rates are observed outside of small neighborhoods of layers ("interior estimates," or "localization results").
   - Interior estimates are impossible for Galerkin. Layers create global pollution.
5. Galerkin/least-squares is conceptually simpler than SUPG and slightly easier to analyze.

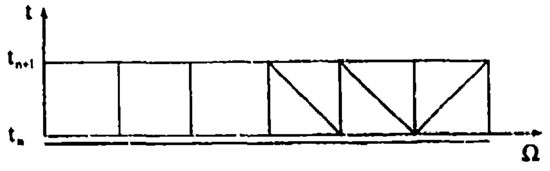
Space-time formulations

Initial-value problem foı ıvection-diffusion equation:

$$\begin{aligned} \mathcal{L}_t u \overset{\text{def}}{=} u_{,t} + \mathcal{L}u &= f && \text{on } \Omega\times]0,T[\\ u(\boldsymbol{x},0) &= u_0(\boldsymbol{x}) && \boldsymbol{x}\in\Omega \\ u &= 0 && \text{on } \Gamma\times]0,T[\end{aligned}$$

Discontinuous Galerkin method in time

Space-time (i.e., $\Omega\times]0,T[$) is divided into *time slabs*, $\Omega\times]t_n,t_{n+1}[$, where $0=t_0<t_1<\cdots<t_N=T$.



$\mathcal{V}^h$ = piecewise polynomials of order k, continuous in $\boldsymbol{x}$, but *discontinuous* across time slabs.

Point of departure: **Galerkin's method**

Find $u^h=u^h(\boldsymbol{x},t)$ such that for all $w^h=w^h(\boldsymbol{x},t)$,

$$B_n(w^h,u^h)=L_n(w^h), \qquad n=0,1,\ldots,N-1$$

where

$$B_n(w^h,u^h) \overset{\text{def}}{=} \int_{t_n}^{t_{n+1}} \left(-\int_\Omega w^h_{,t}\,u^h\,d\Omega + B(w^h,u^h)\right) dt + \int_\Omega w^h(t_{n+1}^-)\,u^h(t_{n+1}^-)\,d\Omega$$

$$L_n(w^h) \overset{\text{def}}{=} \int_{t_n}^{t_{n+1}} L(w^h)\,dt + \int_\Omega w^h(t_n^+)\,u^h(t_n^-)\,d\Omega$$

$$u^h(t_0^-) \overset{\text{def}}{=} u^h(\boldsymbol{x},t_0^-)=u_0(\boldsymbol{x}) \qquad \boldsymbol{x}\in\Omega$$

Remark: Continuity of the solution across time slabs is *weakly* enforced.

Generalization of the other methods proceeds analogously to the steady case. For example,

Galerkin/least-squares

$$B_n(w^h,u^h)+\int_{t_n}^{t_{n+1}}\sum_e\int_{\Omega^e}\mathcal{L}_t w^h\,\tau(\mathcal{L}_t u^h-f)\,d\Omega\,dt=L_n(w^h)$$

Remarks:

1. The mathematical convergence theory is virtually identical to its steady counterpart.
2. Same error estimates and localization results hold in terms of the order, k, of the space-time elements employed.
3. The issue of time integrator is obviated by the choice of space-time interpolation.
4. Unconditional stability is achieved for any choice.
5. Gives rise to a system of linear algebraic equations on each time slab.

Symmetric linear advective-diffusive systems

$$(\mathrm{V}) \qquad \mathcal{L}_t V \overset{\text{def}}{=} A_0 V_{,t} + \mathcal{L}V = \mathcal{F}$$

$$\mathcal{L}V \overset{\text{def}}{=} \widetilde{A} \cdot \nabla V - \nabla \cdot \widetilde{K} \nabla V$$

$$V = (V_1, V_2, \ldots, V_m)^T$$

$$\widetilde{A}^T = \left[\widetilde{A}_1, \widetilde{A}_2, \ldots, \widetilde{A}_d\right]$$

$$\widetilde{K} = \begin{bmatrix} \widetilde{K}_{11} & \cdots & \widetilde{K}_{1d} \\ \vdots & & \vdots \\ \widetilde{K}_{d1} & \cdots & \widetilde{K}_{dd} \end{bmatrix}$$

$$\widetilde{A} \cdot \nabla V = \widetilde{A}^T \nabla V = \widetilde{A}_1 \frac{\partial V}{\partial x_1} + \cdots + \widetilde{A}_d \frac{\partial V}{\partial x_d}$$

$$A_0 = m \times m \text{ symm.}, > 0$$

$$\widetilde{A}_i = m \times m \text{ symm.}, 1 \le i \le d$$

$$\widetilde{K} = (m \cdot d) \times (m \cdot d) \text{ symm.}, \ge 0$$

Find $V = V(x, t)$ satisfying (V) and

$$V(x, 0) = V_0(x) \qquad x \in \Omega$$

$$V = 0 \qquad \text{on } \Gamma$$

Analogous to the scalar, steady case we have:

Galerkin

$$B(W^h, V^h) = L(W^h)$$

$$B(W^h, V^h) \overset{\text{def}}{=} \int_\Omega (-\nabla W^h \cdot \widetilde{A} V^h + \nabla W^h \cdot \widetilde{K} \nabla V^h)\, d\Omega$$

$$L(W^h) \overset{\text{def}}{=} \int_\Omega W^h \cdot \mathcal{F}\, d\Omega$$

Galerkin/least-squares

$$B(W^h, V^h) + \sum_e \int_{\Omega^e} \mathcal{L}W^h\, \tau(\mathcal{L}V^h - \mathcal{F})\, d\Omega = L(W^h)$$

Remarks:

1. $\tau = m \times m$ symm., > 0 (H.-Mallet, 1986).
2. Simple arguments reveal that τ should not be diagonal, even in 1d.

Space-time formulations are developed in identical fashion to the scalar case:

Galerkin

$$B_n(W^h, V^h) = L_n(W^h), \qquad n = 0, 1, \ldots, N-1$$

$$B_n(W^h, V^h) \overset{\text{def}}{=} \int_{t_n}^{t_{n+1}} \left(-\int_\Omega W^h_{,t} \cdot A_0 V^h\, d\Omega + B(W^h, V^h) \right) dt + \int_\Omega W^h(t_{n+1}^-) \cdot A_0 V^h(t_{n+1}^-)\, d\Omega$$

$$L_n(W^h) \overset{\text{def}}{=} \int_{t_n}^{t_{n+1}} L(W^h)\, dt + \int_\Omega W^h(t_n^+) \cdot A_0 V^h(t_n^-)\, d\Omega$$

$$V^h(t_0^-) \overset{\text{def}}{=} V^h(x, t_0^-) = V_0^h(x) \qquad x \in \Omega$$

Galerkin/least-squares

$$B_n(W^h, V^h) + \int_{t_n}^{t_{n+1}} \sum_e \int_{\Omega^e} \mathcal{L}_t W^h \cdot \tau(\mathcal{L}_t V^h - \mathcal{F})\, d\Omega\, dt = L_n(W^h)$$

Remark: Error estimates analogous to those of the scalar case may be proved for Galerkin/least-squares (and SUPG).

A definition of the matrix τ (H.-Mallet 1986, Shakib 1988)

$$\tau = L^{-T} \left(L^{-1} \hat{A}_\xi^T \bar{A}_0^{-1} \hat{A}_\xi L^{-T} \right)^{-1/2} L^{-1}$$

where

$$A_0 = LL^T \quad , \quad \text{Cholesky factorization}$$

$$\bar{A}_0 = \text{block-diag}(A_0, \ldots, A_0)$$

$$\hat{A}_\xi = \begin{bmatrix} \frac{\partial \xi_0}{\partial t} \widetilde{A}_0 \\ \frac{\partial \xi_1}{\partial x_i} \widetilde{A}_i \\ \vdots \\ \frac{\partial \xi_d}{\partial x_i} \widetilde{A}_i \\ \frac{3}{\sqrt{d}} \frac{\partial \xi_1}{\partial x_i} \frac{\partial \xi_1}{\partial x_j} \widetilde{K}_{ij} \\ \vdots \\ \frac{3}{\sqrt{d}} \frac{\partial \xi_d}{\partial x_i} \frac{\partial \xi_d}{\partial x_j} \widetilde{K}_{ij} \end{bmatrix}$$

Remark: If A_0 is only positive *semi*definite, then A_0^{-1} is to be understood as the inverse on the non-degenerate subspace, e.g.

$$\text{if } A_0 = \begin{bmatrix} c & 0 \\ 0 & 0 \end{bmatrix}, \text{ then } A_0^{-1} \overset{\text{def}}{=} \begin{bmatrix} c^{-1} & 0 \\ 0 & 0 \end{bmatrix}$$

Incompressible Euler and Navier-Stokes Equations as a Symmetric Advective-Diffusive System

$$V = \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ p \end{Bmatrix}$$

$$A_0 = \rho \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\widetilde{A}_i = u_i A_0 + \begin{bmatrix} 0 & e_i \\ e_i^T & 0 \end{bmatrix} \qquad 1 \le i \le 3$$

$e_i =$ cartesian basis vector

$$\widetilde{K}_{11} = \widetilde{K}_{22} = \widetilde{K}_{33} = \nu A_0 \quad , \qquad \nu = \frac{\mu}{\rho}$$
$$\widetilde{K}_{ij} = 0 \quad , \quad i \neq j$$

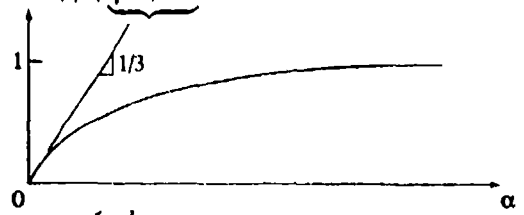
Remarks:

1. $\boldsymbol{\tau}$ is very simple in general. Assuming

$$\frac{\partial \xi_0}{\partial t} = 0 \quad , \quad \frac{\partial \xi_i}{\partial x_j} = \frac{2}{h}\delta_{ij} \quad ,$$

$$\boldsymbol{\tau} = \begin{bmatrix} \tau & 0 & 0 & 0 \\ 0 & \tau & 0 & 0 \\ 0 & 0 & \tau & 0 \\ 0 & 0 & 0 & \lambda \end{bmatrix}$$

$$\tau = \frac{h}{2\rho|\boldsymbol{u}|}\underbrace{\sqrt{\frac{\alpha^2}{9+\alpha^2}}}\ ; \quad \alpha = \frac{h|\boldsymbol{u}|}{2\nu} = \text{element Reynolds number}$$



$$\tau \approx \begin{cases} \dfrac{h}{2\rho|\boldsymbol{u}|} & \text{as } \alpha \to \infty \\ \dfrac{h^2}{12\mu} & \text{as } \alpha \to 0 \end{cases} \quad ; \qquad \lambda = \frac{\rho|\boldsymbol{u}|h}{2}$$

2. The formulation simplifies considerably due to the sparsity of the arrays. See Szepessy, 1987, and Hansbo-Szepessy, 1990, for an analysis and numerical results for a related method at high Reynolds number, and Johnson-Sarinen, 1986, for analysis of incompressible Navier-Stokes via related formulations.

3. Assume $V_{,t} = 0$ and $\widetilde{A}_i = \begin{bmatrix} 0 & e_i \\ e_i^T & 0 \end{bmatrix}$. Then, the method becomes a *nonsymmetric* Stokes solver, and is convergent for all continuous pressure interpolation (i.e., no Babuška-Brezzi condition). Galerkin/least-squares and related approaches to the Stokes problem:
   Brezzi–Pitkäranta, 1984
   H.–Franca–Balestra, 1986
   H.–Franca, 1987
   Brezzi–Douglas, 1988
   Franca–H.–Loula–Miranda, 1988
   Franca–H., 1988
   Pierre, 1988, 1989
   Franca, 1989
   Douglas–Wang, 1989
   Durán–Nochetto, 1989
   Sylvester–Kechkar, 1990
   Stenberg, 1990
   Franca–Stenberg, 1991
   Franca–H.–Stenberg, 1991

Compressible Euler and Navier-Stokes equations

(U) $$U_{,t} + A \cdot \nabla U - \nabla \cdot K \nabla U = \mathcal{F}$$

$$U = (\rho,\ \rho u_1,\ \rho u_2,\ \rho u_3,\ \rho e)^T$$

Remarks:

1. Neither A nor K is symmetric or definite.
2. Classical $L_2(\Omega)$ stability estimates are derived by taking the dot product of (U) with U. This does *not* even make dimensional sense,

$$\frac{1}{2}\frac{d}{dt}\,[\rho^2 \underbrace{(1 + |\mathbf{u}|^2 + e^2)}_{???}\,] + \cdots$$

This suggests that the $L_2(\Omega)$-inner-product structure is inappropriate for compressible Navier-Stokes and consequently so would be a classical type Galerkin formulation.

Entropy variables

Godunov 1962, Mock 1980, Harten 1983, Tadmor 1984, Dutt 1985, H. *et al.* 1986, Johnson *et al.* 1987.

$$H = H(U) = -\rho s$$

$$s = \ln\left(\frac{p}{p_0}\left(\frac{\rho_0}{\rho}\right)^{\gamma}\right) = \text{non-dimensional entropy}$$

$$V = \frac{\partial H}{\partial U} = \text{entropy variables}$$

Remarks:

1. H is a *convex* function of U. Thus $U = U(V)$ is a well-defined change of variables which transforms (U) to (V).
2. The dot product of (V) with V results in the *Clausius-Duhem inequality*:

$$0 = V \cdot (A_0 V_{,t} + \tilde{A} \cdot \nabla V - \nabla \cdot (\widetilde{K} \nabla V) - \mathcal{F})$$

implies

$$(\rho\eta)_{,t} + \nabla \cdot (\rho\eta u) + \nabla \cdot \left(\frac{q}{\theta}\right) + \frac{\rho r}{\theta} \geq 0$$

3. The space-time formulation inherits this property. Replacing W^h by V^h results in a global statement of C.D.I.
4. In practice, the term

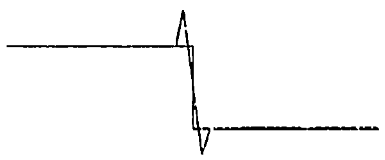
$$W^h \cdot \left(A_0 V^h_{,t} + \tilde{A} \cdot \nabla V^h\right) = W^h \cdot \left(U(V^h)_{,t} + \nabla \cdot F(U(V^h))\right)$$

appearing in $B_n(W^h)$ is integrated-by-parts over each time slab. Global conservation is attained even when approximate element quadrature is employed.

Nonlinear operators and shock-capturing

Remarks:

1. Galerkin/least-squares, SUPG are *linear* methods.
2. They produce approximations to discontinuities like



3. No linear higher-order accurate method will produce *monotone* profiles.
4. The idea is to introduce *nonlinear operators* in order to control oscillations about discontinuities, but not upset higher-order accuracy in smooth regions ("locally first-order"). H.-Mallet-Mizukami 1986, H.-Mallet 1986, Dutra do Carmo-Galeão 1987, Johnson-Szepessy 1986-1989.

$$\int_{t_n}^{t_{n+1}} \sum_e \int_{\Omega^e} \beta \, \hat{\nabla}_\xi W^h \cdot \tilde{A}_0 \hat{\nabla}_\xi V^h \, d\Omega \, dt$$

$\beta = \beta(V^h)$ is a scalar

$\hat{\nabla}_\xi$ = a non-dimensional space-time element gradient operator

$\tilde{A}_0 = \text{block-diag}(A_0, \ldots, A_0)$

Examples:

1. $$\beta = \left|\mathcal{L}_t V^h - \mathcal{F}\right|_\tau \Big/ \left|\tilde{A}_0 \hat{\nabla}_\xi V^h\right|_{\tilde{\tau}}$$

where

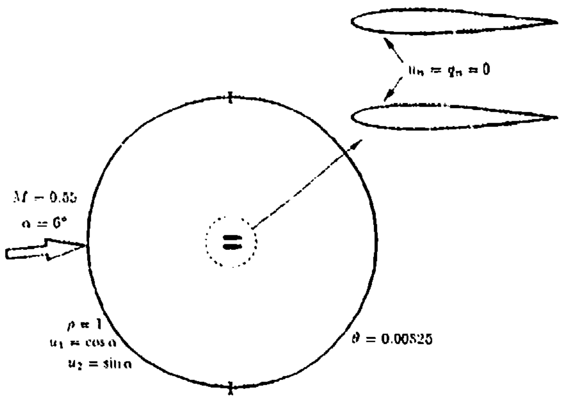
$$\tilde{\tau} = \text{block-diag}(\tau, \ldots, \tau)$$
$$|X|_\tau = (X \cdot \tau X)^{\frac{1}{2}}$$
$$|X|_{\tilde{\tau}} = (X \cdot \tilde{\tau} X)^{\frac{1}{2}}$$

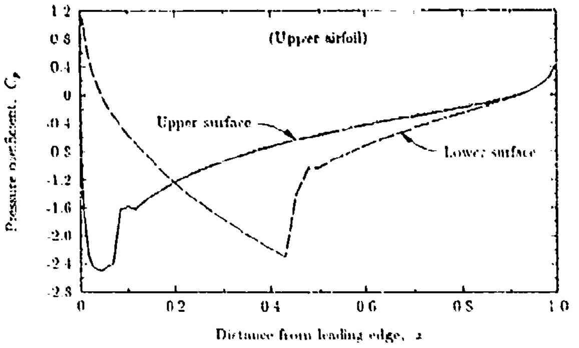
2. $$\beta = 2\left|\mathcal{L}_t V^h - \mathcal{F}\right|_\tau^2 \Big/ \left|\hat{\nabla}_\xi V^h\right|_{A_0}^2$$

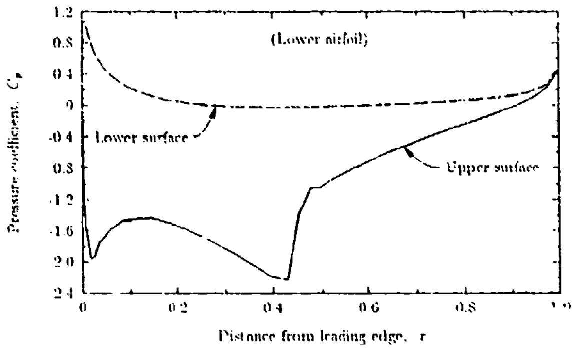
where

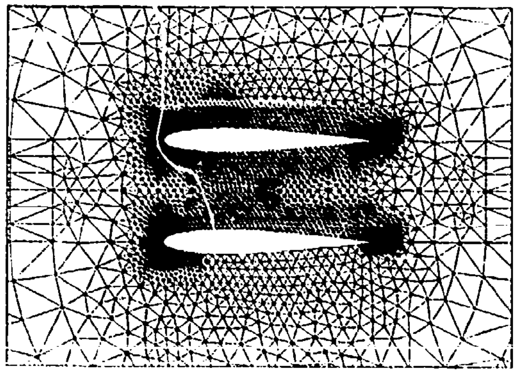
$$|X|_{A_0} = (X \cdot A_0 X)^{\frac{1}{2}}$$

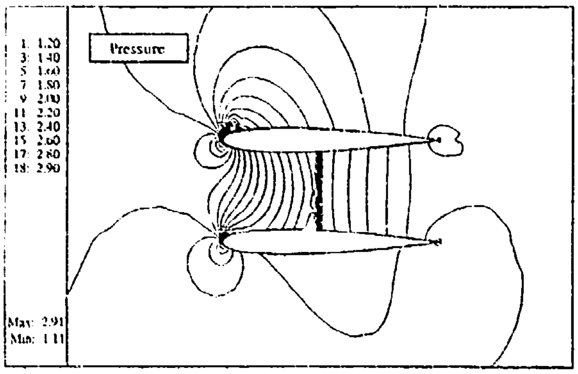
Biaca-0012 Airfoil



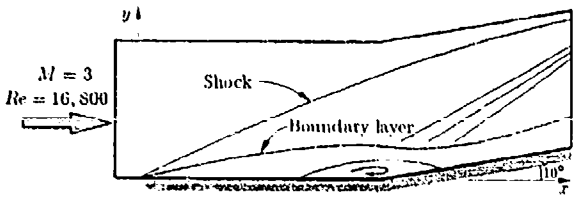




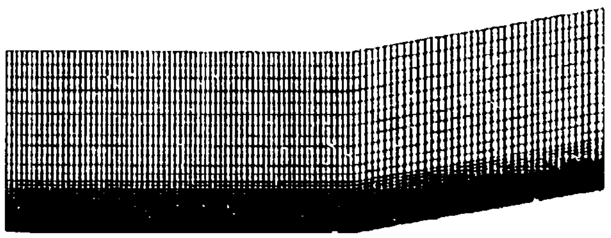


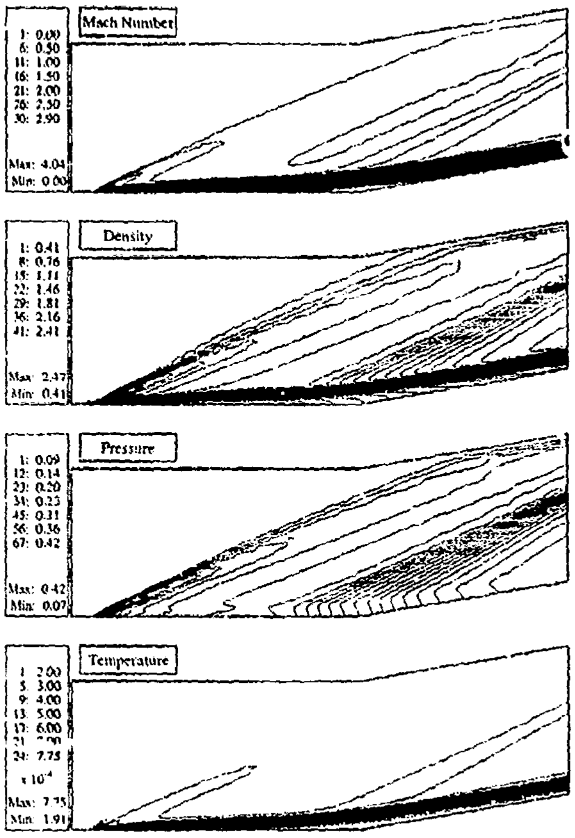


Compression Corner – Mach 3

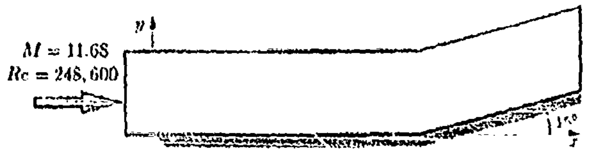


- 4,056 elements; 4200 nodes

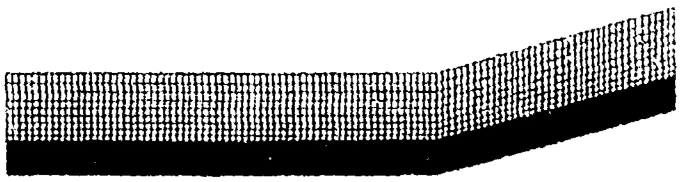


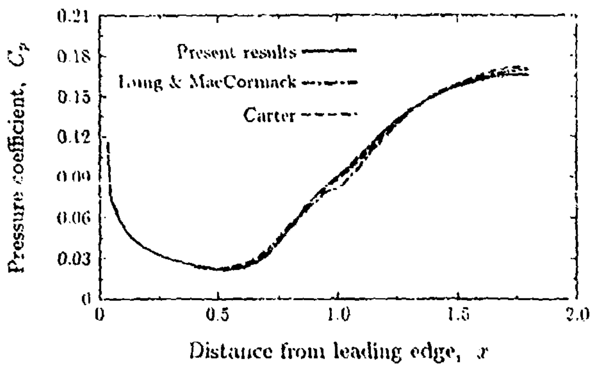


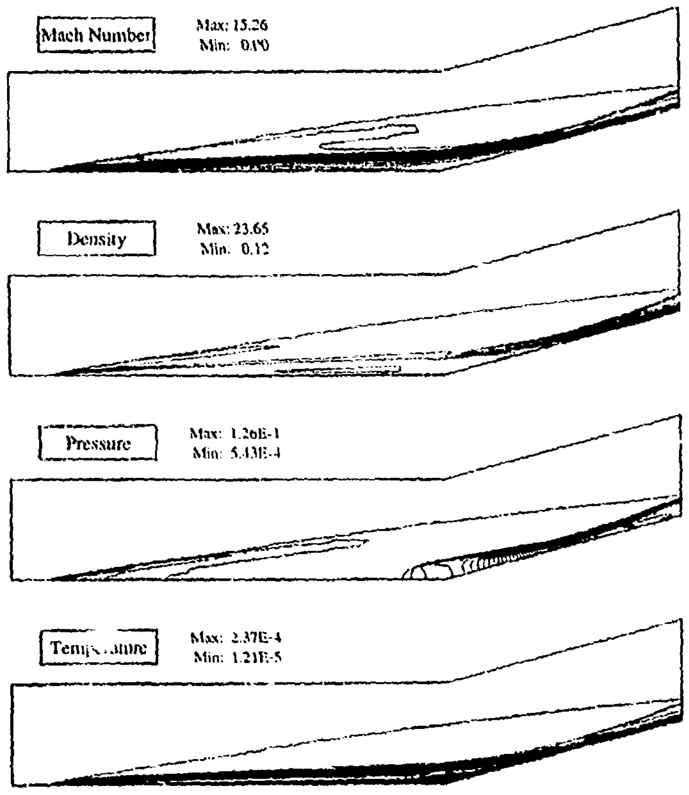
Compression Corner – Mach 11.68

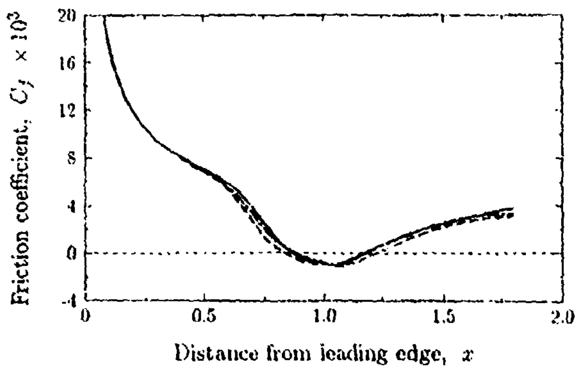


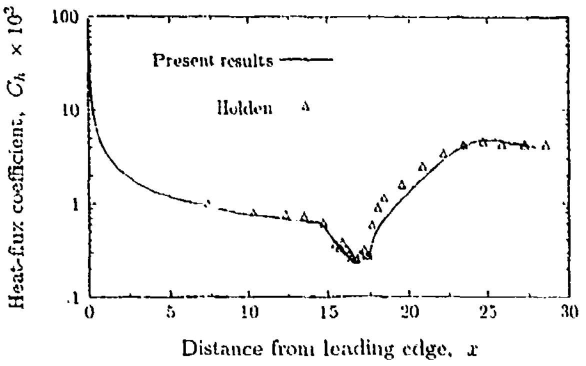
- 14,144 elements; 14,404 nodes

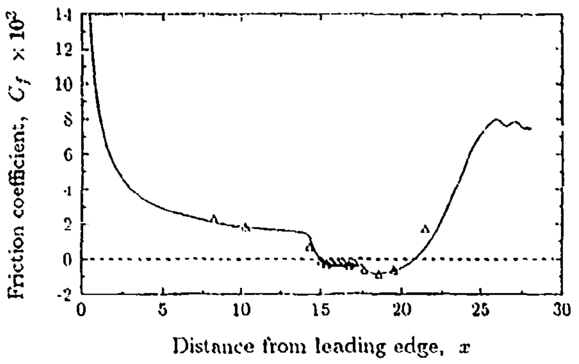




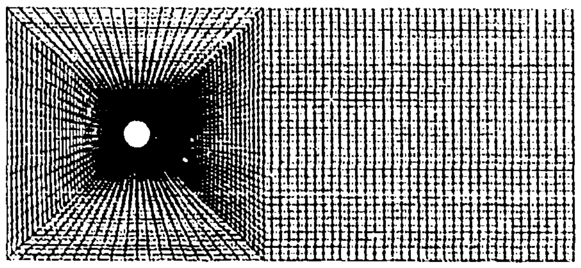


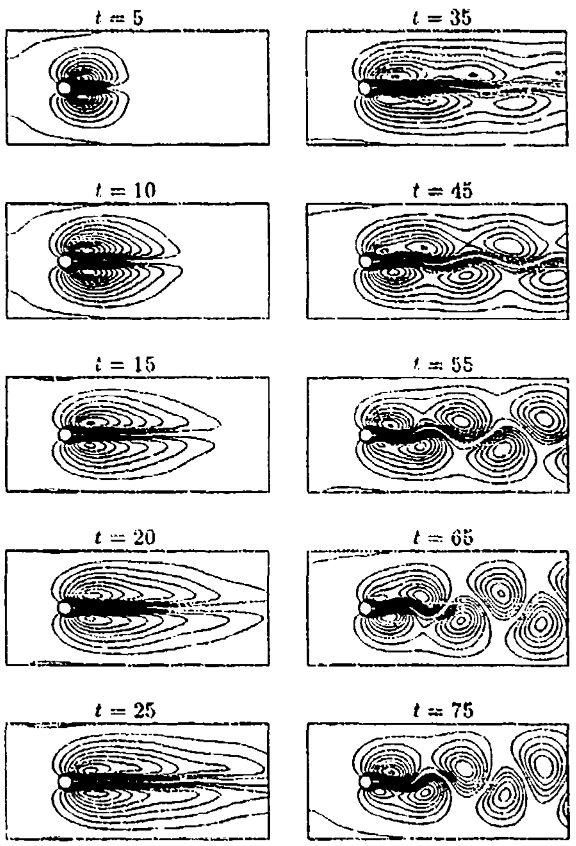






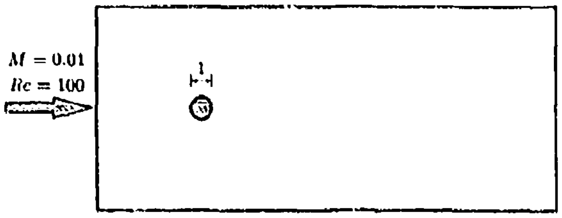
• 4,936 elements; 5,063 nodes



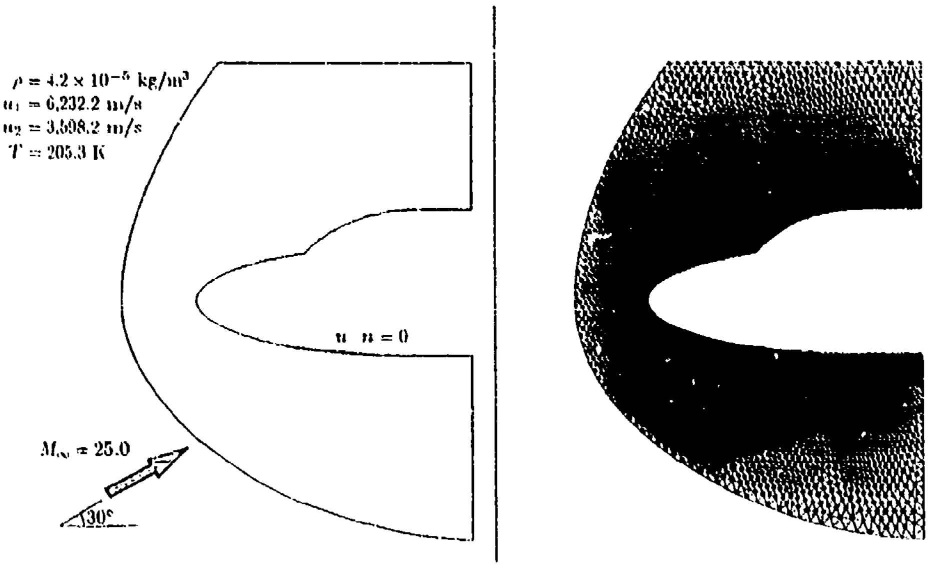


Flow Past a Circular Cylinder

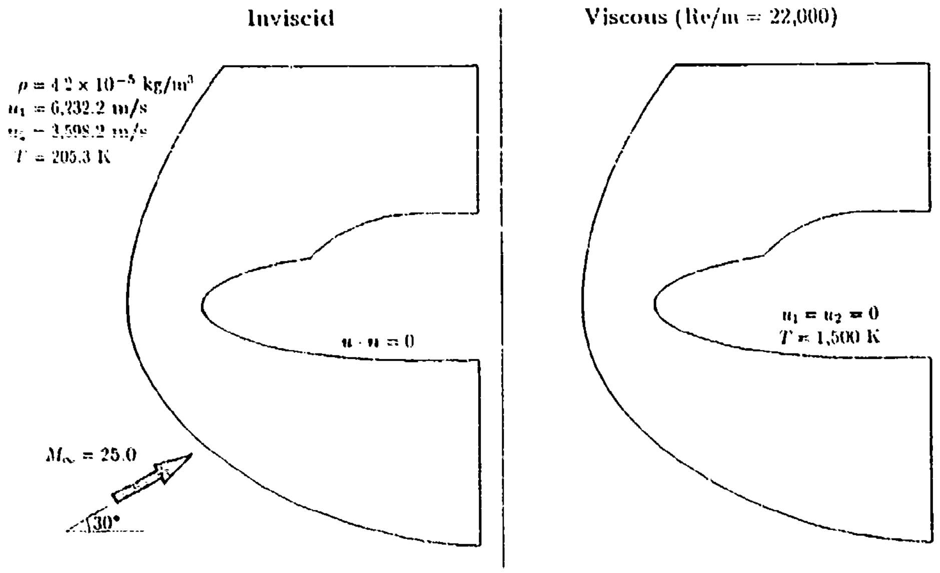
(Nearly-Incompressible)



Flow past a Double Ellipse ($M_\infty = 25.0$)



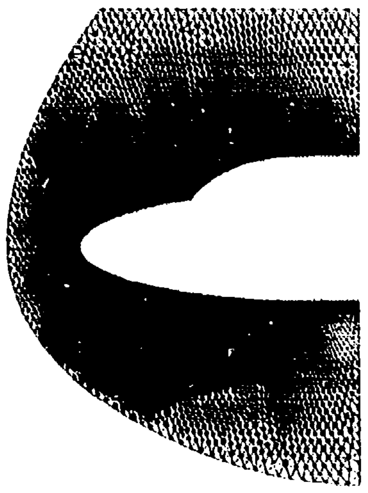
Flow past a Double Ellipse ($M_\infty = 25.0$)

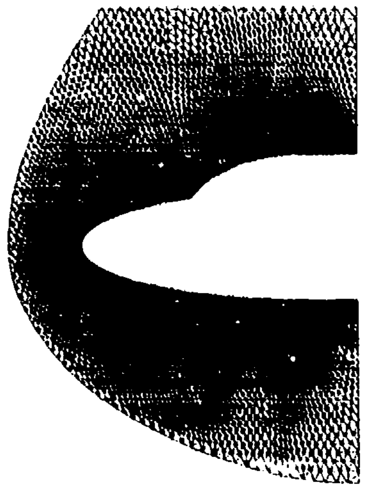


Flow past a Double Ellipse ($M_\infty = 25.0$)

Inviscid

Viscous

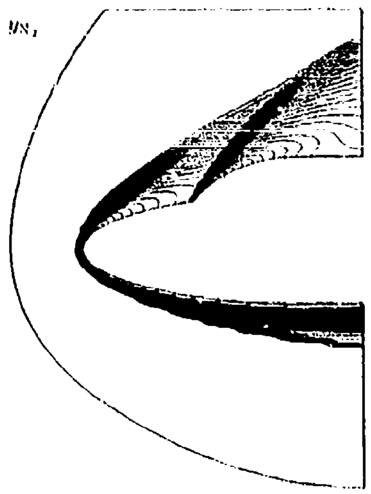


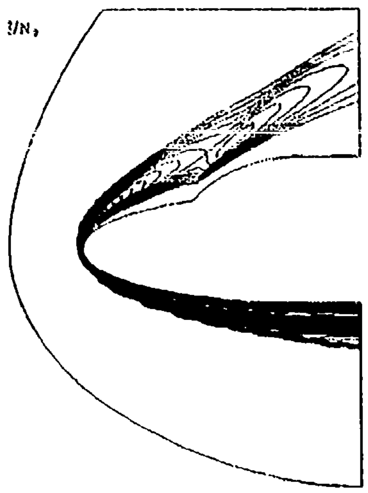


Flow past a Double Ellipse ($M_\infty = 25.0$)

Inviscid

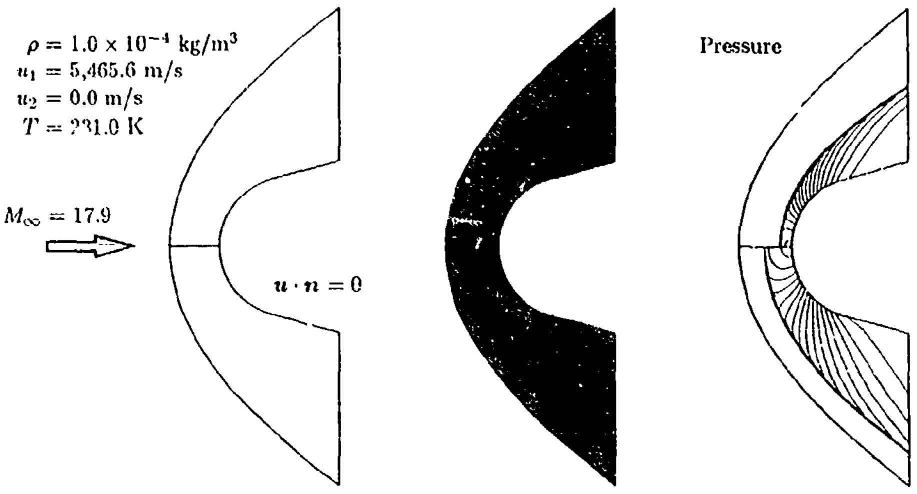
Viscous





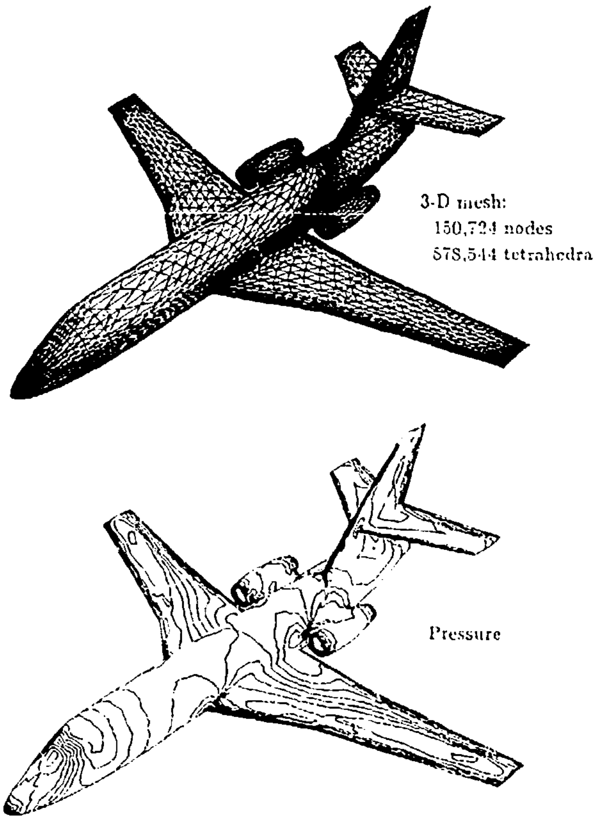
Flow past a Blunt Body ($M_\infty = 17.9$)

Top: equilibrium chemistry – Bottom: perfect gas



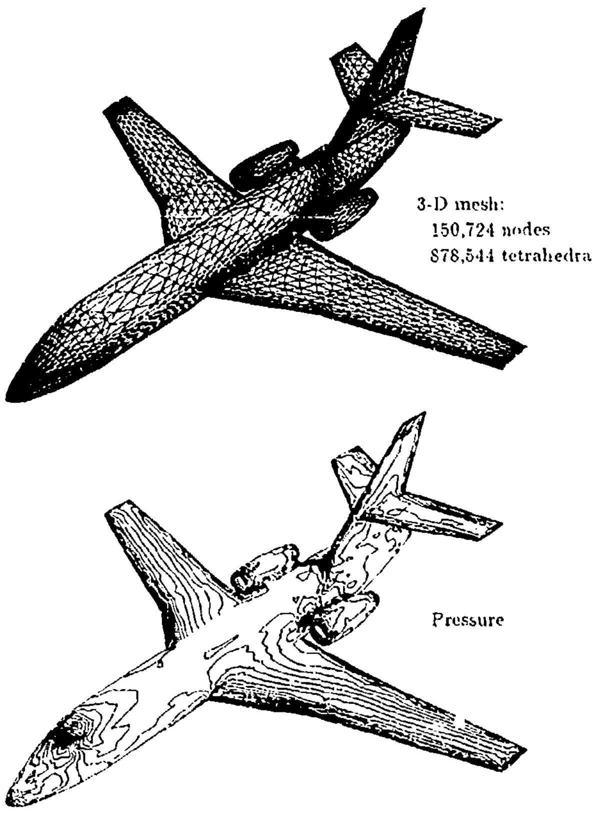
Flow around a Falcon Jet: Cruise configuration

Mach 0.85 – $\alpha = 1°$ – Inviscid flow



Flow around a Falcon Jet: Approach configuration

Mach [illegible] – $\alpha = 6°$ – $\beta = 7.2°$ – Inviscid flow



Conclusions

1. a) Galerkin/least-squares is an effective method for advective-diffusive systems.

   b) Nonlinear shock-capturing operators are essential when discontinuities and unresolved sharp layers are present.

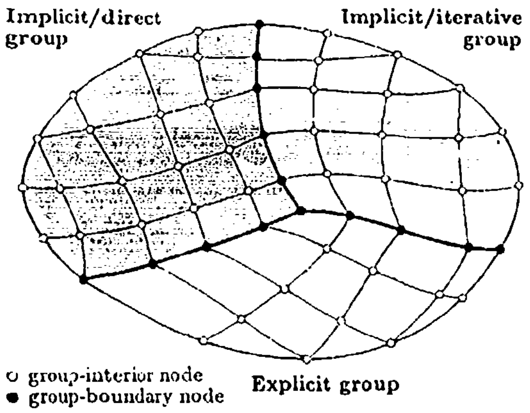
   c) Mathematical convergence proofs have been established for all linear and some nonlinear situations.

2. Focal points of research:
   - Compressible Euler and Navier-Stokes equations.
   - Generation and adaptive refinement of unstructured meshes, especially in three dimensions.
   - Design and analysis of effective shock-capturing operators.

3. Current developments:
   - More attention paid to turbulence modelling.
   - Commercially available compressible flow codes.
   - Application to more complex *systems*, in particular, chemically reacting flows, combustion, MHD.
   - Increased emphasis on iterative strategies for parallel architectures.
   - Further mathematical analysis of algorithms and the development of design principles based upon the mathematical theory of finite elements.

Multi-Element Group Partitioning (Domain Decomposition)

- Algorithm allows different solution techniques on different subdomains
- For example:

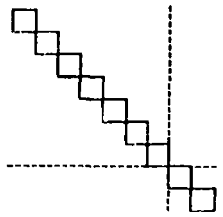


- Generalizable to arbitrary number of element groups

Structure of the Left–Hand–Side Submatrices

- Explicit element group

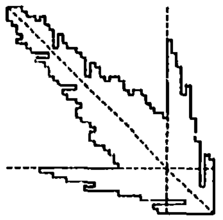
$$A^{exp} = \begin{bmatrix} A_{11}^{exp} & 0 \\ 0 & A_{22}^{exp} \end{bmatrix} =$$



Uncoupled set of symm., pos. def. nodal block matrices

- Implicit/direct element group

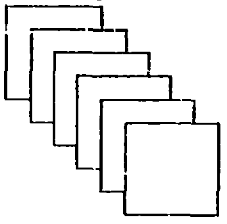
$$A^{dir} = \begin{bmatrix} A_{11}^{dir} & A_{12}^{dir} \\ A_{21}^{dir} & A_{22}^{dir} \end{bmatrix} =$$



Fully coupled – stored in skyline column height form

- Implicit/iterative element group

$$A^{iter} = \begin{bmatrix} A_{11}^{iter} & A_{12}^{iter} \\ A_{21}^{iter} & A_{22}^{iter} \end{bmatrix} =$$



Fully coupled – stored in "unassembled" element file

Global System of Equations

- The left-hand-side matrix

$$\begin{bmatrix} A_{11}^{exp} & 0 & 0 & 0 \\ 0 & A_{11}^{dir} & 0 & A_{12}^{dir} \\ 0 & 0 & A_{11}^{iter} & A_{12}^{iter} \\ 0 & A_{21}^{dir} & A_{21}^{iter} & (A_{22}^{exp} + A_{22}^{dir} + A_{22}^{iter}) \end{bmatrix}$$

- The right-hand-side vector

$$\begin{Bmatrix} b_1^{exp} \\ b_1^{dir} \\ b_1^{iter} \\ (b_2^{exp} + b_2^{dir} + b_2^{iter}) \end{Bmatrix}$$

- The vector of unknowns

$$\begin{Bmatrix} x^{exp} \\ x^{dir} \\ x^{iter} \\ x^{body} \end{Bmatrix}$$

Partial Reduction of the System

- Explicit element group: solve for x^{exp}

$$A_{11}^{exp}\ x^{exp} = b_1^{exp}$$

- Implicit/direct element group: statically condense x^{dir}

$$\hat{A}_{22}^{dir} \overset{\text{def}}{=} A_{22}^{dir} - A_{21}^{dir}(A_{11}^{dir})^{-1}A_{12}^{dir}$$
$$\hat{b}_2^{dir} \overset{\text{def}}{=} b_2^{dir} - A_{21}^{dir}(A_{11}^{dir})^{-1}b_1^{dir}$$

- Reduce the system of equations to

$$\begin{bmatrix} A_{11}^{iter} & A_{12}^{iter} \\ A_{21}^{iter} & (A_{22}^{exp} + \hat{A}_{22}^{dir} + A_{22}^{iter}) \end{bmatrix} \begin{Bmatrix} x^{iter} \\ x^{bndy} \end{Bmatrix} = \begin{Bmatrix} b_1^{iter} \\ (b_2^{exp} + \hat{b}_2^{dir} + b_2^{iter}) \end{Bmatrix}$$

Rewrite as

$$\tilde{A}\ \tilde{x} = \tilde{b}$$

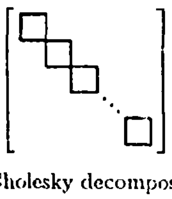
Pre–preconditioning

- Transform system to enhance convergence of iterative solver
- *Diagonal* pre–preconditioning

$$\tilde{W} \overset{\text{def}}{=} \text{diag}(\tilde{A})$$

$$\underbrace{(\tilde{W}^{-\frac{1}{2}}\tilde{A}\tilde{W}^{-\frac{1}{2}})}_{A}\underbrace{(\tilde{W}^{\frac{1}{2}}\tilde{x})}_{x} = \underbrace{(\tilde{W}^{-\frac{1}{2}}\tilde{b})}_{b}$$

- *Block–diagonal* pre–preconditioning

$$\tilde{W} \overset{\text{def}}{=} \text{block}(\tilde{A}) =$$


$$= \tilde{U}^T\tilde{U} \qquad \text{(Cholesky decomposition)}$$

$$\underbrace{(\tilde{U}^{-T}\tilde{A}\tilde{U}^{-1})}_{A}\underbrace{(\tilde{U}\tilde{x})}_{x} = \underbrace{(\tilde{U}^{-T}\tilde{b})}_{b}$$

GMRES Algorithm

(Saad and Shultz – Mallet et al.)

- Search for solution $x = x_0 + z$, such that

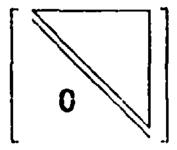
$$\min_{z \in K} \|b - A(x_0 + z)\|$$
$$K \overset{\text{def}}{=} \text{Span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$$
$$r_0 = b - Ax_0$$

- Calculate the orthogonal basis of K,

$$U_k = [u_1, u_2, \ldots, u_k]$$

using Modified Gram–Schmidt orthogonalization

- Generate a rectangular upper Hessenberg matrix, H_k

$$H_k =$$


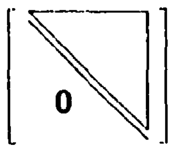
$$(k+1) \times k$$

such that

$$AU_k = U_{k+1}H_k$$

- Reduce the minimization problem to

$$\min_{z \in K} \|b - A(x_0 + z)\| =$$
$$\min_y \|\ e\ -\ H_k\ y\ \| =$$

$$\min_y \left\| \begin{Bmatrix} \|r_0\| \\ 0 \\ \vdots \\ 0 \end{Bmatrix} - \right.$$


$$\left. \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{Bmatrix} \right\|$$

$$z = \sum_{j=1}^{k} y_j u_j$$

- Solve the above problem using a QR algorithm

Preconditioning on Element Group Basis

- Formally replace the system of equations by

$$(L^{-1}AU^{-1})(Ux) = (L^{-1}b)$$

- Preconditioner

$$L \overset{\text{def}}{=} \prod_{g=1}^{n_{eg}} L^{(g)} \quad ; \quad U \overset{\text{def}}{=} \prod_{g=n_{eg}}^{1} U^{(g)}$$

- For implicit/iterative element group:

  Element–by–element (EBE) preconditioners

$$L^{iter} \overset{\text{def}}{=} \prod_{e=1}^{n_{el}} \bar{L}^{e} \quad ; \quad U^{iter} \overset{\text{def}}{=} \prod_{e=n_{el}}^{1} \bar{U}^{e}$$

- Regularized element arrays

$$\bar{A}^{e} = A^{e} - \text{Diag}(A^{e}) + I$$

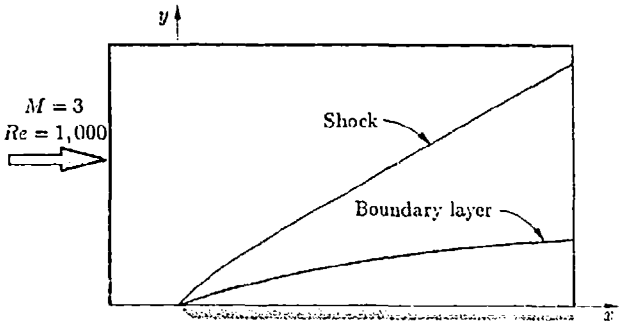
- *Gauss–Seidel EBE* (sum decomposition)

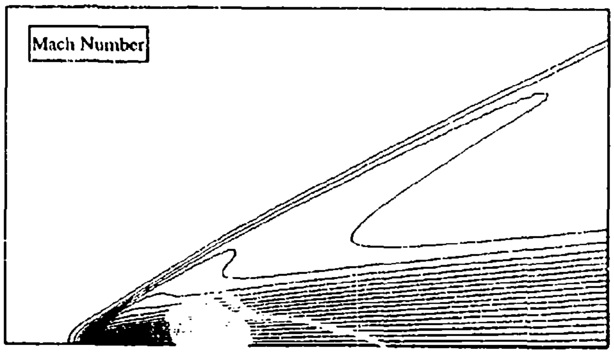
$$\bar{L}^{e} + \bar{U}^{e} = \bar{A}^{e} + I$$

- *Nonsymmetric "Cholesky" EBE* (product decomposition)

$$\bar{L}^{e}\bar{U}^{e} = \bar{A}^{e}$$

Supersonic Flow over a Flat Plate





- CFL = 25

- CPU–time (sec) per linear solve

Elements	Direct	Diagonal	Block–diag.
112	1.5	1.2	0.3
448	12.2	6.1	0.9
1,792	102.7	35.6	5.0
7,168	1,174.4	175.3	24.5
28,672	—	628.3	130.9

Computed on Convex–C1

- Storage of iterative solver is substantially less than direct solver

- Effect of EBE preconditioning

- CPU–time (sec) per linear solve

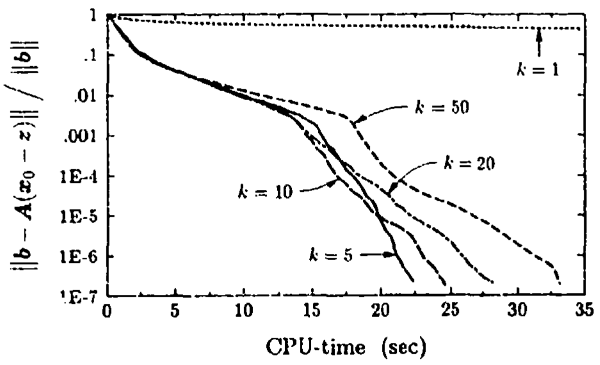
  Block–diagonal pre–preconditioning

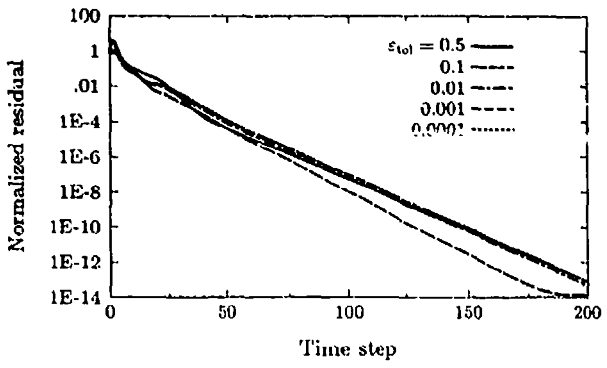
Elements	No-EBE	GS–EBE	NC–EBE
112	0.3	0.3	0.4
448	0.9	0.9	1.3
1,792	5.0	4.8	6.5
7,168	24.5	21.2	30.4
28,672	130.9	112.0	263.0

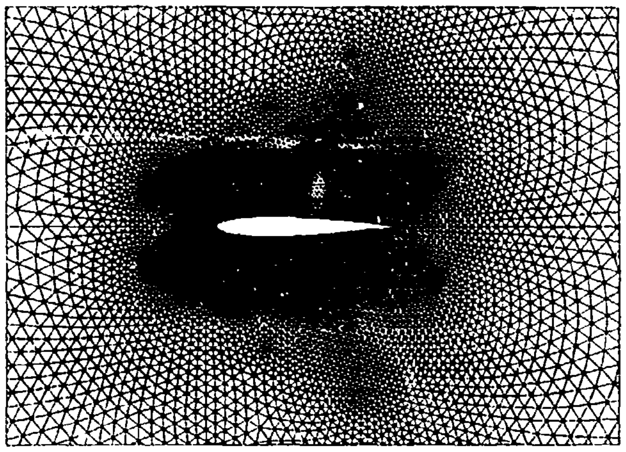
- CFL = 25

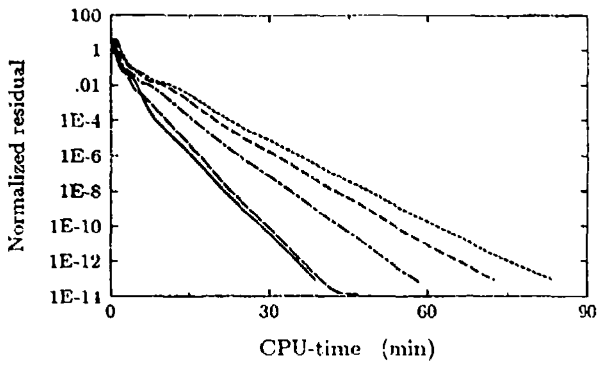
  1,792 elements

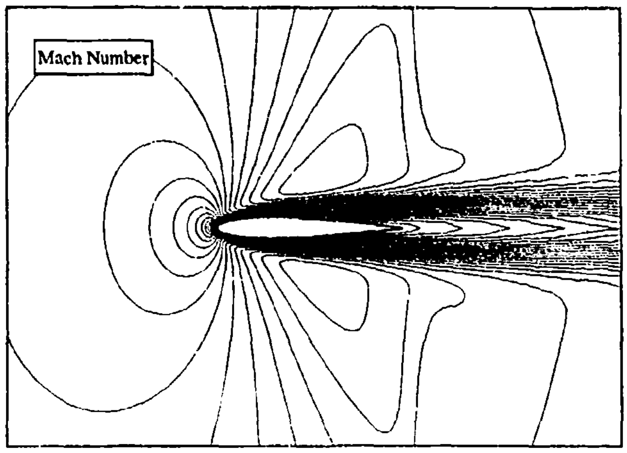
  Block–diagonal pre–preconditioning



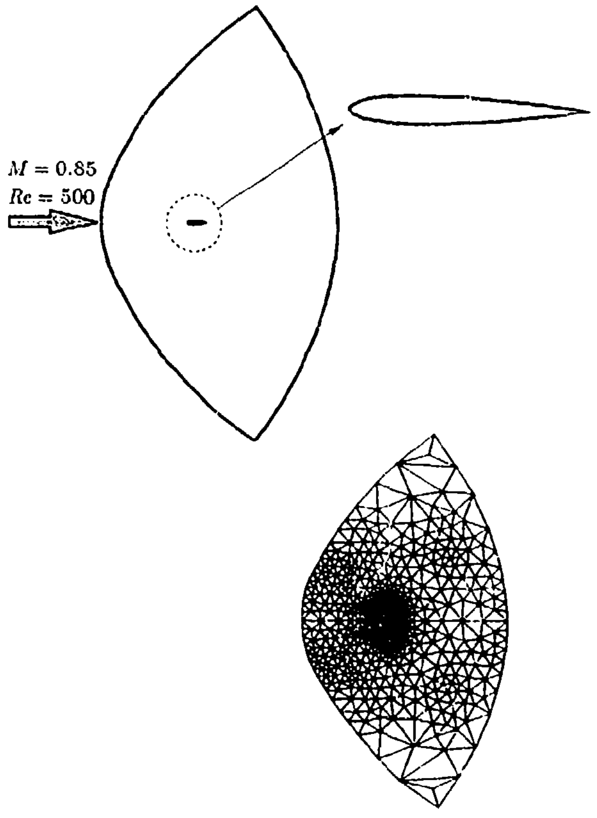




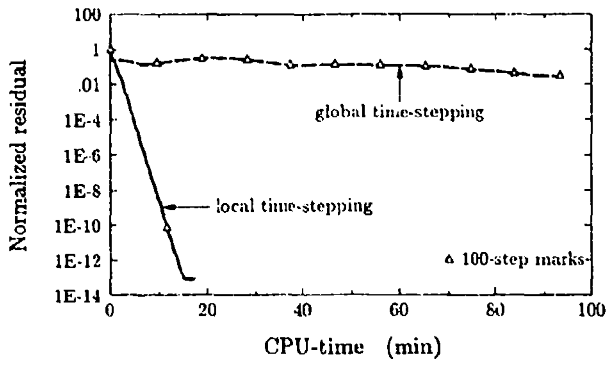




NACA0012 Airfoil



- Comparison of global and local time–stepping strategies
- CFL = 50
- Block–diagonal pre–preconditioning

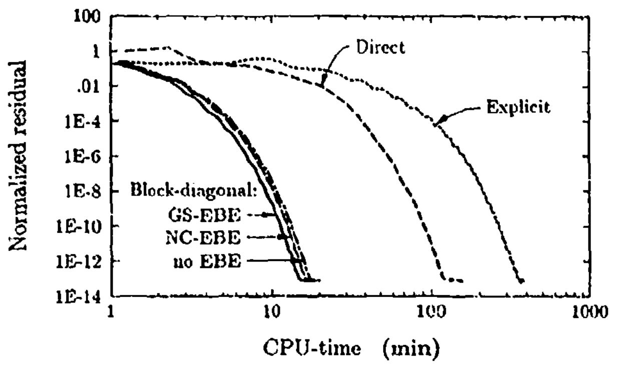


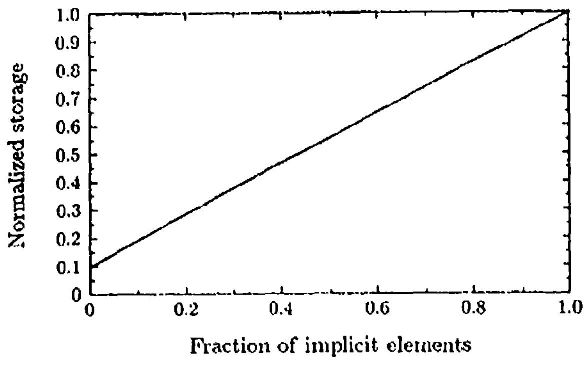
- **Local time stepping strategy**

  CFL = 50 for implicit

  CFL = 0.5 for explicit

  Block–diagonal pre–preconditioning with iterative algorithms

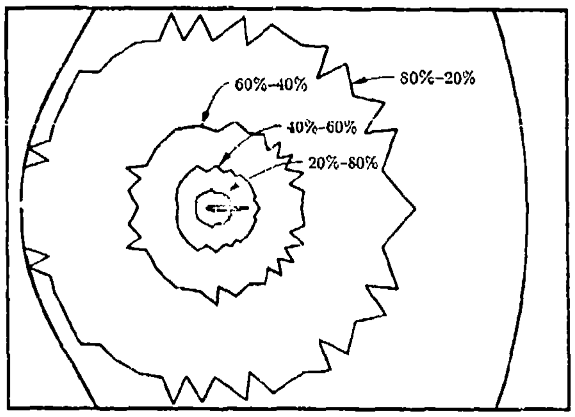


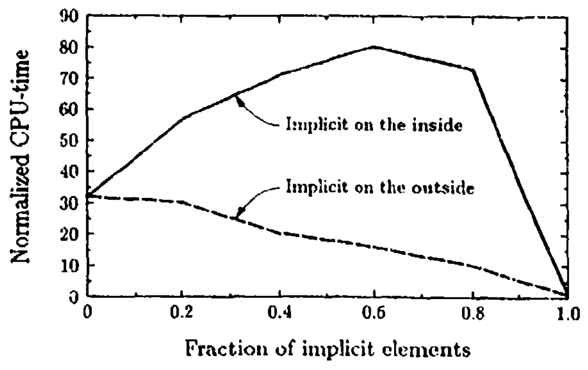


- At normalized residual 10^{-13}:

Method	Time Steps	CPU-Time (min)	Storage (MByte)
Explicit	13,977	356.33	0.830
Direct	104	118.05	9.752
No-EBE	147	17.42	4.249
GS-EBE	132	14.86	4.249
NC-EBE	132	16.53	7.354

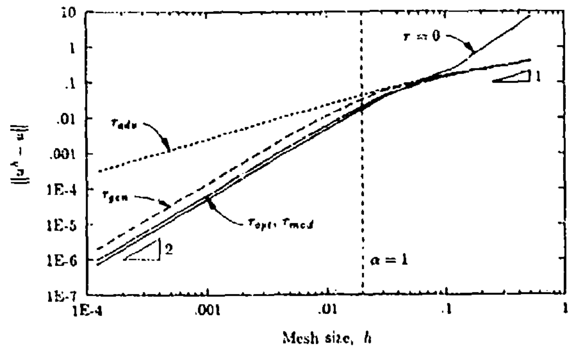
- Two-element group partitions

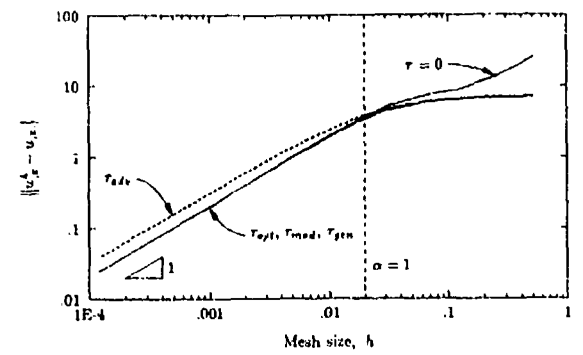


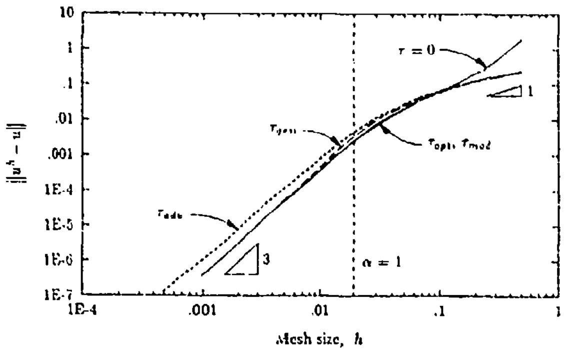


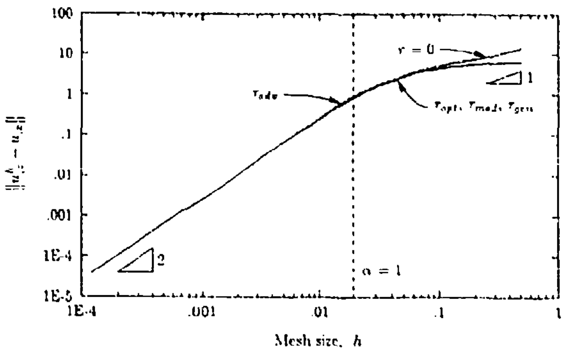
Conclusions

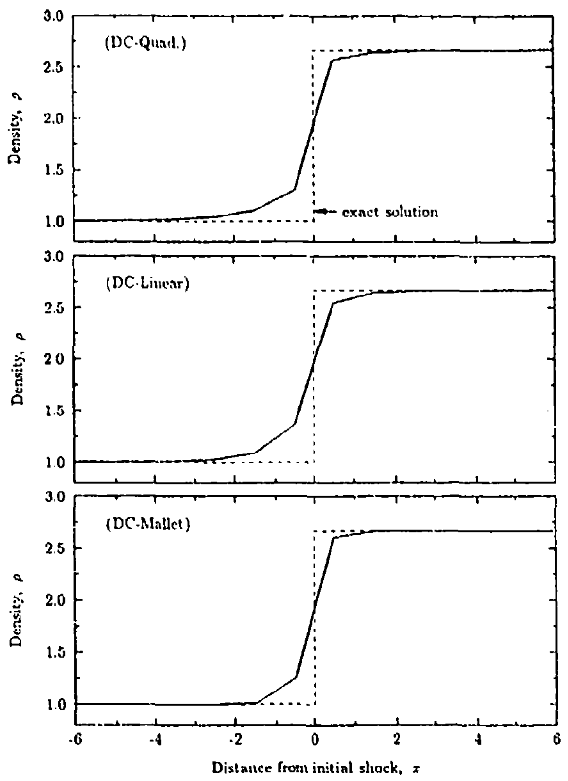
- Block–diagonal pre–preconditioning significantly improves the convergence of the GMRES algorithm.
- Implicit/iterative algorithm is superior to both explicit and implicit/direct algorithm, even for very small problems.
- Implicit/explicit partitioning is useful for making optimal use of storage and CPU resources.
- We expect implicit/iterative algorithm to exhibit even better performance on large three–dimensional problems, but further research in reducing storage of the element arrays is needed.

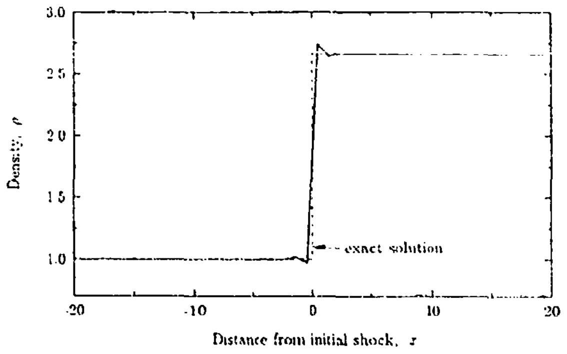


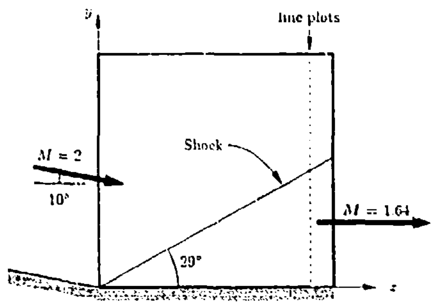


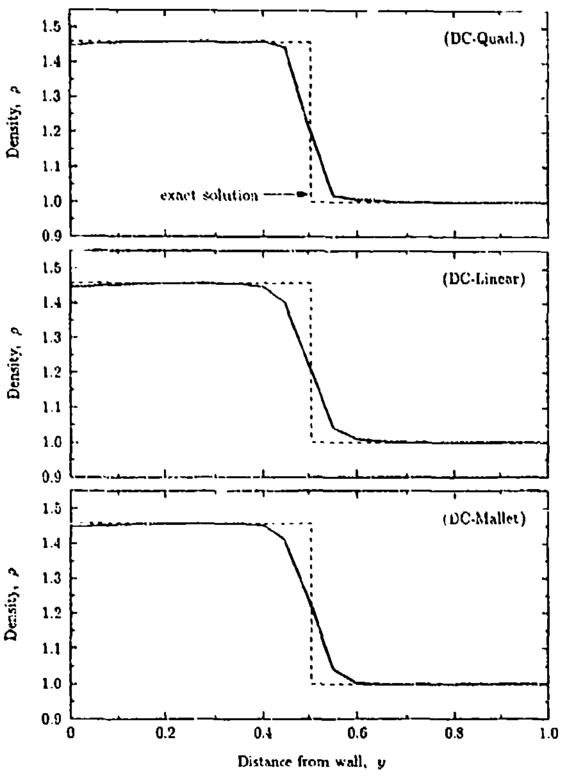

$\|u^h - u\|$
Mesh size, h
$\tau = 0$
τ_{gen}
τ_{opt}, τ_{mod}
τ_{adv}
3
1
$\alpha = 1$

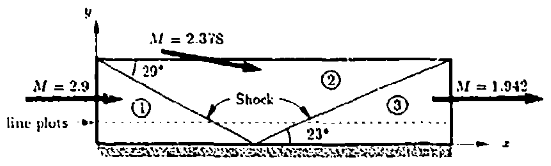

$\|u_{,x}^h - u_{,x}\|$
Mesh size, h
$\tau = 0$
τ_{adv}
$\tau_{opt}, \tau_{mod}, \tau_{gen}$
2
1
$\alpha = 1$

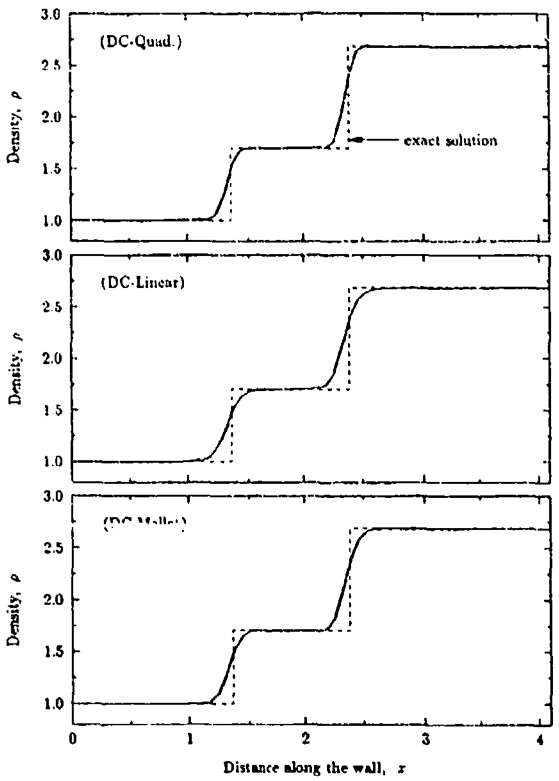

(DC-Quad.)
(DC-Linear)
(DC-Mallet)
Density, ρ
exact solution
Distance from initial shock, x

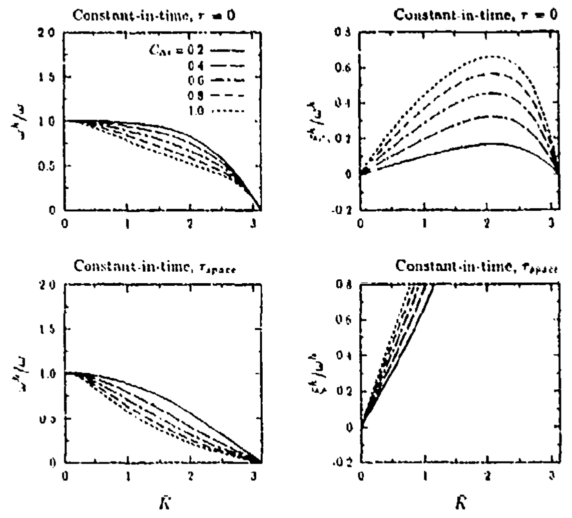

Density, ρ
exact solution
Distance from initial shock, x

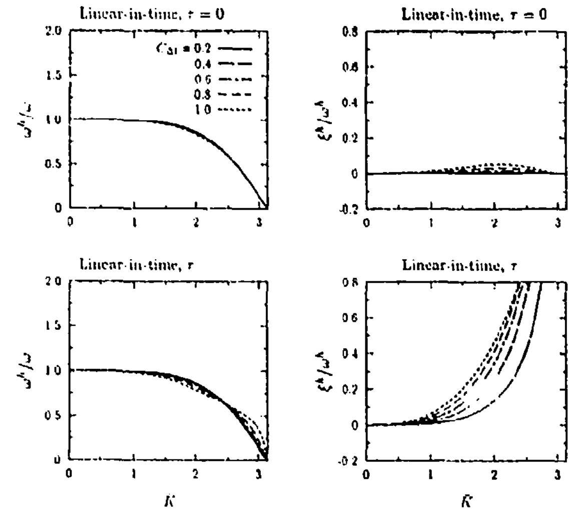

y
line plots
Shock
$M = 2$
10°
$M = 1.64$
29°
x

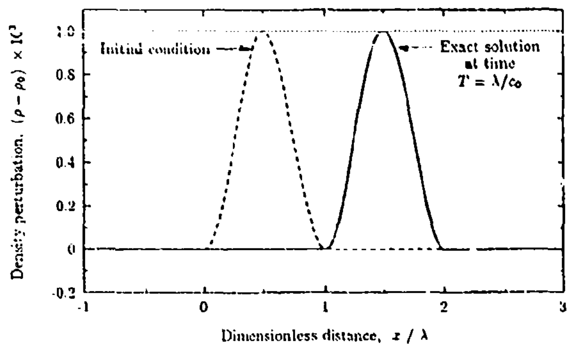

(DC-Quad.)
(DC-Linear)
(DC-Mallet)
exact solution
Density, ρ
Distance from wall, y

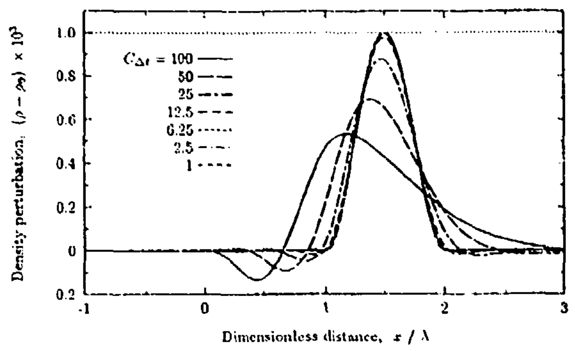

y
M = 2.378
29°
M = 2.9
①
②
③
Shock
M = 1.942
line plots
23°
x


(DC-Quad.)
(DC-Linear)
(DC-Mallet)
exact solution
Density, ρ
Distance along the wall, x


Constant-in-time, τ = 0
Constant-in-time, τ = 0
Constant-in-time, τ_space
Constant-in-time, τ_space
C_Δt = 0.2
0.4
0.6
0.8
1.0
ω^h/ω
ξ^h/ω^h
K̃
K̃


Linear-in-time, τ = 0
Linear-in-time, τ = 0
Linear-in-time, τ
Linear-in-time, τ
C_Δt = 0.2
0.4
0.6
0.8
1.0
ω^h/ω
ξ^h/ω^h
K̃
K̃


Initial condition
Exact solution at time T = λ/c₀
Density perturbation, (ρ − ρ₀) × 10³
Dimensionless distance, x / λ


C_Δt = 100
50
25
12.5
6.25
2.5
1
Density perturbation, (ρ − ρ₀) × 10³
Dimensionless distance, x / λ

References

1. C.I. Bajer, "Notes on the stability of non-rectangular space-time finite elements", *Int. J. Numer. Methods Eng.*, 24, 1721-1739 (1987).

2. A.N. Brooks and T.J.R. Hughes, "Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations", *Comput. Methods Appl. Mech. Eng.*, 32, 199-259 (1982).

3. J.E. Carter, "Numerical solutions of the Navier-Stokes equations for the supersonic laminar flow over a two-dimensional compression corner", *NASA Technical Report*, NASA TR R-385, 1972.

4. F. Chalot, L.P. Franca, I. Harari, T.J.R. Hughes, F. Shakib, M. Mallet, J. Periaux and B. Stoufflet, "Calculation of two-dimensional Euler flows with a new Petrov-Galerkin finite element method", in A. Dervieux and B. Van Leer (eds), *Notes on Numerical Fluid Mechanics*, Vieweg, to appear.

5. P.K. Dutt, "Stable boundary conditions and difference schemes for Navier-Stokes type equations", *Ph.D. Thesis*, University of California, Los Angeles, 1985.

6. R.M. Ferencz, "Element-by-element preconditioning techniques for large-scale, vectorized finite element analysis in nonlinear solid and structural mechanics", *Ph.D. Thesis*, Division of Applied Mechanics, Stanford University, in preparation.

7. L.P. Franca, I. Harari, T.J.R. Hughes, M. Mallet, F. Shakib, T.E. Spelce, F. Chalot and T.E. Tezduyar, "A Petrov-Galerkin finite element method for the compressible Euler and Navier-Stokes equations", in T.E. Tezduyar and T.J.R. Hughes (eds), *Numerical Methods for Compressible Flows - Finite Difference, Element and Volume Techniques, AMD Vol. 78*, ASME, New York, 1986, pp. 19-43.

8. A.C. Galeão and E.G. Dutra do Carmo, "A consistent approximate upwind Petrov-Galerkin method for convection-dominated problems", *Comput. Methods Appl. Mech. Eng.*, 68, 83-95 (1988).

9. S.K. Godunov, "The problem of a generalized solution in the theory of quasilinear equations and in gas dynamics", *Russ. Math. Surveys*, 17, 145-156 (1962).

10. G.H. Golub and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, 1983.

11. B. Gustafsson and A. Sundstrom, "Incompletely parabolic problems in fluid dynamics", *SIAM J. Appl. Math.*, 35, No. 2, pp. 343-357 (1978).

12. A. Harten, "On the symmetric form of system of conservation laws with entropy", *J. Comput. Phys.*, 49, 151-164 (1983).

13. M.S. Holden, "A study of flow separation in regions of shock wave-boundary layer interaction in hypersonic flow", *AIAA 11th Fluid and Plasma Dynamics Conference*, Seattle, Wash., July 10-12, 1978.

14. T.J.R. Hughes, "Recent progress in the development and understanding of SUPG methods with special reference to the compressible Euler and Navier-Stokes equations", *Int. J. Numer. Methods Fluids*, 7, 1261-1275 (1987).

15. T.J.R. Hughes and A.N. Brooks, "A multidimensional upwind scheme with no crosswind diffusion", in T.J.R. Hughes (ed), *Finite Element Methods for Convection Dominated Flows, AMD Vol. 34*, ASME, New York, 1979, pp. 19-35.

16. T.J.R. Hughes and A.N. Brooks, "A theoretical framework for Petrov-Galerkin methods with discontinuous weighting functions. Application to the streamline upwind procedure", in R.H. Gallagher *et al.* (eds), *Finite Element in Fluids, Vol. 4*, Wiley, Chichester 1982, pp. 47-65.

17. T.J.R. Hughes and R.M. Ferencz, "Implicit solution of large-scale contact and impact problems employing an EBE preconditioned iterative solver", *IMPACT 87 International Conference on Effects of Fast Transient Loading in the Context of Structural Mechanics*, Lausanne, Switzerland, August 26-27, 1987.

18. T.J.R. Hughes, R.M. Ferencz and J.O. Hallquist, "Large-scale vectorized implicit calculations in solid mechanics on a CRAY X-MP/48 utilizing EBE preconditioned conjugate gradients", *Comput. Methods Appl. Mech. Eng.*, 61, 215-248 (1987).

19. T.J.R. Hughes, L.P. Franca, I. Harari, M. Mallet, F. Shakib and T.E. Spelce, "Finite element method for high-speed flows: Consistent calculation of boundary flux", *AIAA 25th Aerospace Sciences Meeting, Paper No. 87-0556*, Reno, Nevada, January 1987.

20. T.J.R. Hughes, L.P. Franca and G.M. Hulbert, "A new finite element formulation for computational fluid dynamics: VIII. The Galerkin/least-squares method for advective-diffusive equations", *Comput. Methods Appl. Mech. Eng.*, to appear.

21. T.J.R. Hughes, L.P. Franca and M. Mallet, "A new finite element formulation for computational fluid dynamics: I. Symmetric forms of the compressible Euler and Navier-Stokes equations and the second law of thermodynamics", *Comput. Methods Appl. Mech. Eng.*, 54, 223-234 (1986).

22. T.J.R. Hughes, L.P. Franca and M. Mallet, "A new finite element formulation for computational fluid dynamics: VI. Convergence analysis of the generalized SUPG formulation for linear time-dependent multidimensional advective-diffusive systems", *Comput. Methods Appl. Mech. Eng.*, 63, 97-112 (1987).

23. T.J.R. Hughes and G.M. Hulbert, "Space-time finite element methods for elastodynamics: Formulations and error estimates", *Comput. Methods Appl. Mech. Eng.*, 66, 339-363 (1988).

24. T.J.R. Hughes and M. Mallet, "A new finite element formulation for computational fluid dynamics: III. The generalized streamline operator for multidimensional advective-diffusive systems", *Comput. Methods Appl. Mech. Eng.*, 58, 305-328 (1986).

25. T.J.R. Hughes and M. Mallet, "A new finite element formulation for computational fluid dynamics: IV. A discontinuity-capturing operator for multidimensional advective-diffusive systems", *Comput. Methods Appl. Mech. Eng.*, 58, 329-339 (1986).

26. T.J.R. Hughes, M. Mallet and A. Mizukami, "A new finite element formulation for computational fluid dynamics: II. Beyond SUPG", *Comput. Methods Appl. Mech. Eng.*, 54, 341-355 (1986).

27. T.J.R. Hughes and F. Shakib, "Computational aerodynamics and the finite element method", *AIAA 26th Aerospace Sciences Meeting, Paper No. 88-0031*, Reno, Nevada, January 1988.

28. T.J.R. Hughes and T.E. Tezduyar, "Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations", *Comput. Methods Appl. Mech. Eng.*, 45, 217-284 (1984).

29. T.J.R. Hughes and T.E. Tezduyar, "Analysis of some fully-discrete algorithms for the one-dimensional heat equation", *Int. J. Numer. Methods Eng.*, 21, 163-168 (1985).

30. G.M. Hulbert, "Space-time finite element methods for second-order hyperbolic equations", *Ph.D. Thesis*, Stanford University, in preparation.

31. C.M. Hung and R.W. MacCormack, "Numerical solutions of supersonic and hypersonic laminar compression corner flows", *AIAA Journal*, 14, No. 4, 475-481 (1976).

32. C. Johnson, U. Nävert and J. Pitkäranta, "Finite element methods for linear hyperbolic problems", *Comput. Methods Appl. Mech. Eng.*, 45, 285-312 (1984).

33. C. Johnson and A. Szepessy, "Convergence of a finite element method for a nonlinear hyperbolic conservation law", *Technical Report 1985-25*, Mathematics Department, Chalmers University of Technology, Goteborg, Sweden, 1985.

34. C. Johnson and A. Szepessy, "A shock-capturing streamline diffusion finite element method for a nonlinear hyperbolic conservation law", *Technical Report 1986-09*, Mathematics Department, Chalmers University of Technology, Göteborg, Sweden, 1986.

35. C. Johnson and A. Szepessy, "On the convergence of streamline diffusion finite element methods for hyperbolic conservation laws", in T.E. Tezduyar and T.J.R. Hughes (eds), *Numerical Methods for Compressible Flows - Finite Difference, Element and Volume Techniques, AMD Vol. 78*, ASME, New York, 1986, pp. 75-91.

36. C. Johnson, A. Szepessy and P. Hansbo, "On the convergence of shock capturing streamline diffusion finite element methods for hyperbolic conservation laws", *Technical Report 1987-21*, Mathematics Department, Chalmers University of Technology, Goteborg, 1987.

37. S.K. Jordan and J.E. Fromm, "Oscillatory drag, lift, and torque on a circular cylinder in a uniform flow", *Phys. Fluids*, 15, No. 3, 371-376 (1972).

38. S. Lang, *Differential Manifolds*, Addison-Wesley, Reading, Massachusetts, 1972.

39. M. Mallet, "A finite element method for computational fluid dynamics", *Ph.D. Thesis*, Stanford University, 1985.

40. M. Mallet, J. Periaux and B. Stoufflet, "Convergence acceleration of finite element methods for the solution of the Euler and Navier-Stokes equations of compressible flow", *Proceedings of the 7th GAMM Conference on Numerical Methods in Fluid Dynamics*, to appear.

41. J.E. Marsden and T.J.R. Hughes, *Mathematical Foundations of Elasticity*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

42. A. Mizukami, "A mixed finite element method for boundary flux computation", *Comput Methods Appl Mech Eng*, **57**, 239-243 (1986).

43. M.S. Mock, "Systems of conservation laws of mixed type", *J. Differential Equations*, **37**, 70-88 (1980).

44. B. Nour-Omid, B.N. Parlett and A. Raefsky, "Comparison of Lanczos with conjugate gradient using element preconditioning", in R. Glowinski *et al.* (eds), *Proceedings of the First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, SIAM, Philadelphia, 1988, pp. 250-260.

45. Y. Saad and M.H. Schultz, "GMRES: A generalized minimum residual algorithm for solving nonsymmetric linear systems", *Research Report YALEU/DCS/RR-254*, Yale University, Department of Computer Science, New Haven, 1983.

46. F. Shakib, T.J.R. Hughes and Z. Johan, "A multi-element group preconditioned GMRES algorithm for nonsymmetric systems arising in finite element analysis", *Comput. Methods Appl. Mech. Eng.*, to appear.

47. A. Szepessy, "Convergence of a shock-capturing streamline diffusion finite element method for a scalar conservation law in two space dimensions", *Technical Report 1988-07*, Mathematics Department, Chalmers University of Technology, Göteborg, Sweden, 1988.

48. E. Tadmor, "Skew-selfadjoint forms for systems for conservation laws", *J. Math. Anal Appl*, **103**, 428-442 (1984).

49. P.A. Thompson, *Compressible-Fluid Dynamics*, McGraw-Hill, New York, 1972.

50. E.A. Thornton, P. Dechaumphai and G. Vemaganti, "A finite element approach for prediction of aerothermal loads", *AIAA/ASME 4th Fluid Mechanics, Plasma Dynamics and Lasers Conference, Paper No. 86-1050*, Atlanta, Georgia, May 12-14, 1986.

51. R.F. Warming, R.M. Beam and B.J. Hyett, "Diagonalization and simultaneous symmetrization of the gas-dynamics matrices", *Math. Comput.*, **29**, No. 132, 1037-1045 (1975).

52. R.F. Warming and B.J. Hyett, "The modified equation approach to the stability and accuracy analysis of finite-difference methods", *J. Comput. Physics*, **14**, pp. 159-179 (1974).

FINITE ELEMENT COMPUTATION OF UNSTEADY INCOMPRESSIBLE FLOWS INVOLVING MOVING BOUNDARIES AND INTERFACES AND ITERATIVE SOLUTION STRATEGIES

by

Tayfun E. Tezduyar
Department of Aerospace Engineering and Mechanics,
Army High-Performance Computing Research Center
and
Minnesota Supercomputer Institute
1200 Washington Avenue South
University of Minnesota
Minneapolis, MN 55415
United States

Outline

Abstract

In these lecture notes, we review some of the recent progress on stabilized finite element formulations used in computation of incompressible flows. These stabilization techniques are used to prevent the numerical oscillations that might be generated by the presence of dominant advection terms or by inappropriate combinations of interpolation functions used for the velocity and pressure. The stabilization techniques emphasized in these lecture notes are the Galerkin/least-squares, streamline-upwind/Petrov-Galerkin, and pressure-stabilizing/Petrov-Galerkin formulations, all of them are consistent formulations in the sense that an exact solution still satisfies the stabilized formulation. Some of these techniques are based on finite element discretization in both space and time. Most of the numerical examples considered are unsteady flow problems, with emphasis on those involving moving boundaries and interfaces, such as free-surface flows, liquid drops, flow past an oscillating cylinder and flow past an oscillating airfoil. Flow past a vertically oscillating cylinder mounted on springs is solved as a simple but fundamental fluid-structure interaction problem.

Also reviewed are the iteration strategies employed to solve the implicit equation systems resulting from the finite element discretization of these flow problems, including those discretized by using the space-time formulation. In the space-time formulation the finite element interpolation functions are discontinuous in time so that the fully discrete equations are solved one space-time slab at a time, and this makes the computations feasible. Still, the computational cost associated with the space-time finite element formulations using piecewise linear functions in time is quite heavy. For large-scale problems it becomes imperative to employ efficient iteration methods to reduce the cost involved. This is achieved by using the generalized minimal residual (GMRES) iteration algorithm with the clustered element-by-

element (CEBE) preconditioners. The CEBE preconditioning is a generalized version of the standard element-by-element (EBE) preconditioning. In the CEBE preconditioning the elements are partitioned into clusters of elements, with a desired number of elements in each cluster, and the iterations are performed in a cluster-by-cluster fashion. The number of clusters should be viewed as an optimization parameter to minimize the computational cost. Recently we have been performing this type of computations in the massively parallel environments of the Connection Machines 2 and 5. For these implementations we have been so far using diagonal preconditioners. We will include some examples from these massively parallel computations.

In these lecture notes we also describe a new mixed CEBE/CC preconditioning method for finite element computations. The CC (cluster companion) preconditioning method shares a common philosophy with the multi-grid methods. The CC preconditioners are based on companion meshes associated with different levels of clustering. For each level of clustering, we construct a CEBE preconditioner and an associated CC preconditioner. Because these two preconditioners in a sense complement each other, when they are used in a mixed way, they can be expected to give better performance. In fact, our numerical tests, for two- and three-dimensional problems governed by the Poisson equation, demonstrate that the mixed CEBE/CC preconditioning results in convergence rates which are, in most cases, significantly better than the convergence rates obtained with the best of the CEBE and CC preconditioning methods.

I. Introduction

The purpose of these lecture notes is to present a review of our solution strategies for incompressible flows using finite elements. These strategies include the stabilized formulations, the space-time finite element approach to flow problems with moving boundaries and interfaces, iteration techniques for solving the implicit equation systems involved, massively parallel implementations, and sophisticated preconditioning techniques. The description of the strategies reviewed in these lecture notes, and the numerical results reported, have mostly been extracted from recent articles by Tezduyar et al.[1-5], Mittal and Tezduyar[6] and Behr et al.[7]. The numerical examples considered here are unsteady flow problems, including those involving moving boundaries and interfaces, such as large-amplitude sloshing, liquid drops, flow past an oscillating cylinder and flow past an oscillating airfoil.

Finite element computation of incompressible flows involves two main sources of potential numerical instabilities associated with the Galerkin formulation of a problem. One source is due to the presence of advection terms in the governing equations, and can result in spurious node-to-node oscillations primarily in the velocity field. Such oscillations become more apparent for advection-dominated (i.e., high Reynolds number) flows and flows with sharp layers in the solution. The other source of instability is due to using inappropriate combinations of interpolation functions to represent the velocity and pressure fields. These instabilities usually appear as oscillations primarily in the pressure field. In fact, there is not much about either of these numerical instabilities that could be considered to be inherent to the finite element formulation. Such instabilities appear also in the standard versions of other discretization techniques such as finite difference and finite volume methods.

For the formulations considered in these lecture notes, the stabilization of the numerical method is achieved by adding to the Galerkin formulation a series of stabilizing terms. These terms can be obtained by minimizing the sum of the squared residual of the governing equations integrated over each element domain. This kind of a stabilization is known as the GLS (Galerkin/Least-squares) stabilization. This approach has been successfully applied to Stokes flows[8], compressible flows[9-10], and incompressible flows at finite Reynolds numbers[2-3,11-12]. For time-dependent problems, a strict implementation of the GLS stabilization technique necessitates finite element discretization in both space and time, and therefore leads to a space-time finite element formulation of the problem. The space-time finite element formulation has recently been successfully used, in conjunction with the GLS stabilization, for various problems with fixed spatial domains. We can give as example the work of Hughes et al.[13], Hughes and Hulbert[14], Shakib[10], and Hansbo and Szepessy[12].

Perhaps one of the most striking applications of the stabilized space-time finite element formulation is, as it was first pointed out and implemented by Tezduyar et al.[2-3], in computing moving boundaries and interfaces. The DSD/ST (Deforming-Spatial-Domain/Space-Time) procedure introduced by Tezduyar et al.[2-3] serves this purpose and was successfully applied to several unsteady incompressible flow problems involving moving boundaries and interfaces, such as free-surface flows, liquid drops, two-liquid flows, flows with drifting cylinders. In the DSD/ST procedure the finite element formulation of a problem is written over its space-time domain, and therefore the deformation of the spatial domain with respect to time is taken into account automatically. Furthermore, in the DSD/ST procedure the frequency of remeshing is minimized. Here we define remeshing as the process of generating a new mesh, and projecting the solution from the old mesh to the new one. Since remeshing, in general, involves projection errors, minimizing the frequency of remeshing results in minimizing the projection errors. Furthermore, minimizing the frequency of remeshing increases the massive parallelization potential of the computations.

It is important to realize that the finite element interpolation functions are discontinuous in time so that the fully discrete equations are solved one space-time slab at a time, and this makes the computations feasible. Still, the computational cost associated with the space-time finite element formulations using piecewise linear functions in time is quite heavy. For large-scale problems it becomes imperative to employ efficient iteration methods to reduce the cost involved. This was achieved in Liou and Tezduyar[11] by using the generalized minimal residual (GMRES)[15] iteration algorithm with the clustered element-by-element (CEBE) preconditioners. We will review such preconditioning techniques later in these lecture notes.

Computation of time-dependent incompressible flow problems, over fixed spatial domains, can be performed by using the finite element discretization in space only, rather than in both space and time. In this case we first consider the GLS stabilization for the steady-state equations of incompressible flows. Then in the definition of the stabilizing terms, we replace the residual of the steady-state equations with the time-dependent ones. These stabilizing terms are added to the Galerkin formulation of the time-dependent equations. If, at the element interiors, we further neglect the contribution to the weighting function from the viscous terms (it is identically zero for linear velocity interpolation) we get a formulation with combination of SUPG (streamline-upwind/Petrov-Galerkin) and PSPG (pressure-stabilizing/Petrov-Galerkin) stabilizations. The former prevents the numerical oscillations caused by the presence of advection terms, while the latter allows one to use equal-order functions for velocity and pressure without generating oscillations in the pressure.

The SUPG formulation was introduced by Hughes and Brooks[16]. A comprehensive description of the formulation, together with various numerical examples, can be found in Brooks and Hughes[17]. The implementation of the SUPG formulation in Brooks and Hughes[17] was based on Q1P0 (bilinear velocity/constant pressure) elements and one-step time-integration of the semi-discrete equations obtained by using such elements. For hyperbolic systems in general, and compressible Euler equations in particular, the SUPG stabilization was first reported by Tezduyar and Hughes[18]. The SUPG stabilization for the vorticity-stream function formulation of incompressible flow problems, including those with multiply-connected domains, was introduced by Tezduyar et al.[19].

It was shown that (see Brezzi and Pitkaranta[20], and Hughes et al.[21]), with proper stabilization, elements which do not satisfy the Brezzi condition can be used for Stokes flow problems. The Petrov-Galerkin stabilization proposed in Hughes et al.[21] is achieved, just like in the SUPG stabilization, by adding to the Galerkin formulation a series of integrals over element domains. The PSPG stabilization term proposed in Tezduyar et al.[1] is a generalization, to finite Reynolds number flows, of the Petrov-Galerkin stabilization term proposed in Hughes et al.[21] for Stokes flows. In Tezduyar et al.[1], the SUPG and PSPG stabilizations are used together with both one-step (T1) and multi-step (T6) time-integration schemes[22]. With the T1 scheme, the SUPG and PSPG stabilizations are applied simultaneously. With the T6 scheme, on the other hand, the SUPG stabilization is applied only to the steps involving the advective terms, and the PSPG stabilization is applied only to the steps involving the pressure terms. Both schemes were implemented in Tezduyar et al.[1] based on the Q1Q1 (bilinear velocity and pressure) and P1P1 (linear velocity and pressure) elements, and were successfully applied to a set of nearly standard test problems.

The element-by-element (EBE) preconditioners, which are constructed as series products of element level matrices, have been successfully applied to several classes of problems[23-26]. They can be used effectively with the conjugate-gradient and GMRES[15] methods, and are highly vectorizable and parallelizable[25,27-28]. They can also be used together with the implicit-explicit and adaptive implicit-explicit time-integration schemes[26,28-30]. In CEBE (clustered element-by-element) preconditioning[11,31], the elements are merged into clusters of elements, and the preconditioners are constructed as series products of cluster level matrices. In Liou and Tezduyar[31], the CEBE preconditioning, together with the conjugate-gradient method, was used for solving problems with symmetric spatial operators (e.g., for problems governed by the Poisson equation). In Liou and Tezduyar[11], the CEBE preconditioning was employed, in conjunction with the GMRES method, to solve compressible and incompressible flow problems. Applications to the space-time finite element formulation of incompressible flows were included in Liou and Tezduyar[11]. To facilitate vectorization and parallel processing, as it is done in the grouped element-by-element (GEBE) method[27], the clusters can be grouped in such a way that no two clusters in any group are connected. Furthermore, depending on the number of elements in the cluster, within each cluster, elements can again be grouped in the same way. Each cluster matrix is formed by assembling together the element level matrices associated with the elements in that cluster. The number of elements in each cluster can be viewed as an optimization parameter that can be varied to minimize the computational cost. In fact, in Mittal and Tezduyar[6], the unsteady incompressible flow computations were performed by using a space-time finite element formulation with a nearly optimal cluster size which was determined by numerical experimentation.

In these lecture notes we review the CC (cluster companion) preconditioning introduced by Tezduyar et al.[5]. In the construction process of the CC preconditioners, we first start with a "primary" mesh with different levels of clustering. For each level of clustering in this primary mesh, we define a "companion" mesh, such that each cluster of the primary mesh forms an element of the companion mesh. We then define a CC preconditioner based on each companion mesh, such that there is a CC preconditioner associated with each CEBE preconditioner based on a certain level of clustering. This way, for each level of clustering, we obtain a CC preconditioner which we expect to have more inter-cluster coupling information then the associated CEBE preconditioner has. Conversely, the CEBE preconditioner can be expected to have more intra-cluster coupling information than the associated CC preconditioner has.

The mixed CEBE/CC preconditioning introduced by Tezduyar et al.[5] is based on the belief that the CEBE and CC preconditioners complement each other, and therefore when they are mixed together they will result in better convergence rates. The mixed preconditioning can be implemented by using these two preconditioners alternately at each iteration of the conjugate gradient method or at each outer iteration of the GMRES method. Recently Saad[32] has formulated a new version of the GMRES algorithm which allows changing the preconditioner at every inner iteration. In fact, a GMRES subroutine, based on this new formulation and made available to us by Saad, is what we use to implement our mixed preconditioning.

II. The Governing Equations of Unsteady Incompressible Flows

Let $\Omega_t \in \mathbb{R}^{n_{sd}}$ be the spatial domain at time $t \in (0,T)$, where n_{sd} is the number of space dimensions. Let Γ_t denote the boundary of Ω_t. We consider the following velocity-pressure formulation of the Navier-Stokes equations governing unsteady incompressible flows:

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) - \nabla \cdot \boldsymbol{\sigma} = 0 \qquad \text{on } \Omega_t \ \forall t \in (0,T), \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \qquad \text{on } \Omega_t \ \forall t \in (0,T), \tag{2}$$

where ρ and $\mathbf{u}$ are the density and velocity, and σ is the stress tensor given as

$$\boldsymbol{\sigma}(p, \mathbf{u}) = -p\mathbf{I} + 2\mu\,\boldsymbol{\varepsilon}(\mathbf{u}) \tag{3}$$

with

$$\boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right). \tag{4}$$

Here p and μ are the pressure and the dynamic viscosity, and $\mathbf{I}$ is the identity tensor. The part of the boundary at which the velocity is assumed to be specified is denoted by $(\Gamma_t)_g$:

$$\mathbf{u} = \boldsymbol{g} \qquad \text{on } (\Gamma_t)_g \ \forall t \in (0,T). \tag{5}$$

The "natural" boundary conditions associated with (1) are the conditions on the stress components, and these are the conditions assumed to be imposed at the remaining part of the boundary:

$$\mathbf{n} \cdot \boldsymbol{\sigma} = \boldsymbol{h} \qquad \text{on } (\Gamma_t)_h \ \forall t \in (0,T). \tag{6}$$

The homogeneous version of (6), which corresponds to the "traction-free" (i.e., zero normal and shear stress) conditions, is often imposed at the out flow boundaries. As initial condition, a divergence-free velocity field $\mathbf{u}_0(\mathbf{x})$ is specified over the domain Ω_t at $t = 0$:

$$\mathbf{u}(\mathbf{x},0) = \mathbf{u}_0(\mathbf{x}) \qquad \text{on } \Omega_0. \tag{7}$$

Let us now consider two immiscible fluids, A and B, occupying the domain Ω_t. Let $(\Omega_t)_A$ denote the subdomain occupied by fluid A, and $(\Gamma_t)_A$ denote the boundary of this subdomain. Similarly, let $(\Omega_t)_B$ and $(\Gamma_t)_B$ be the subdomain and boundary associated with fluid B. Furthermore, let $(\Gamma_t)_{AB}$ be the intersection of $(\Gamma_t)_A$ and $(\Gamma_t)_B$, i.e., the interface between fluids A and B.

The kinematical conditions at the interface $(\Gamma_t)_{AB}$ are based on the continuity of the velocity field. The dynamical conditions at the interface, for two-dimensional problems, can be expressed by the following equation:

$$\mathbf{n}_A \cdot \sigma_A + \mathbf{n}_B \cdot \sigma_B = \mathbf{n}_A \ \gamma \ / R_A \qquad \text{on } (\Gamma_t)_{AB} \ \forall \ t \in (0,T) \ , \tag{8}$$

where $\mathbf{n}_A$ and $\mathbf{n}_B$ are the unit outward normal vectors at the interface, σ_A and σ_B are the stress tensors, γ is the surface tension coefficient, and R_A is the radius of curvature defined to be positive when $\mathbf{n}_A$ points towards the center of curvature. The condition (8) is applicable also to free-surface flows (i.e., when the second fluid does not exist), provided that subdomain $(\Omega_t)_A$ is the one assigned to be occupied by the fluid.

III. The Space-Time Formulation with the Galerkin/Least-squares Stabilization and Application to Moving Boundaries and Interfaces: the DSD/ST Procedure

A. The method

In the space-time finite element formulation, the time interval $(0,T)$ is partitioned into subintervals $I_n = (t_n, t_{n+1})$, where t_n and t_{n+1} belong to an ordered series of time levels $0 = t_0 < t_1 < \ldots < t_N = T$. It was first shown in Tezduyar et al.[2,3] that the stabilized space-time finite element formulation can be effectively applied to fluid dynamics computations involving moving boundaries and interfaces. In this formulation the spatial domains at various time levels are allowed to vary. We let $\Omega_n = \Omega_{t_n}$ and $\Gamma_n = \Gamma_{t_n}$, and define the space-time slab Q_n as the space-time domain enclosed by the surfaces Ω_n, Ω_{n+1} and P_n (see Figure 1). Here P_n, the lateral surface of Q_n, is the surface described by the boundary Γ, as t traverses I_n. Similar to the way it was represented by equations (5) and (6), P_n is decomposed into $(P_n)_g$ and $(P_n)_h$ with respect to the type of boundary condition being imposed.

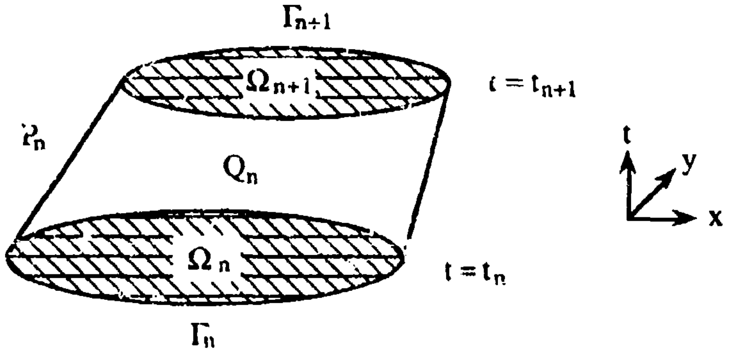


Figure 1. The space-time slab for the DSD/ST formulation

Finite element discretization of a space-time slab Q_n is achieved by dividing it into elements Q_n^e, $e=1,2,\ldots,(n_{el})_n$, where $(n_{el})_n$ is the number of elements in the space-time slab Q_n. Associated with this discretization, for each space-time slab we define the following finite element interpolation function spaces for the velocity and pressure:

$$(S_u^h)_n = \{ \mathbf{u}^h \mid \mathbf{u}^h \in [H^{1h}(Q_n)]^{n_{sd}}, \mathbf{u}^h \doteq \mathbf{g}^h \text{ on } (P_n)_g \} , \tag{9}$$

$$(V_u^h)_n = \{ \mathbf{w}^h \mid \mathbf{w}^h \in [H^{1h}(Q_n)]^{n_{sd}}, \mathbf{w}^h \doteq \mathbf{0} \text{ on } (P_n)_g \} , \tag{10}$$

$$(S_p^h)_n = (V_p^h)_n = \{ q^h \mid q^h \in H^{1h}(Q_n) \} . \tag{11}$$

Here $H^{1h}(Q_n)$ represents the finite-dimensional function space over the space-time slab Q_n. This space is formed by using, over the parent (element) domains, first-order polynomials in space and time. It is also possible to use zeroth-order polynomials in time. In either case, globally, the interpolation functions are continuous in space but discontinuous in time.

The space-time formulation of (1)-(8) can be written as follows: start with

$$(\mathbf{u}^h)_0^- = (\mathbf{u}_0)^h \ ; \tag{12}$$

sequentially for $Q_1, Q_2, \ldots, Q_{N-1}$, given $(\mathbf{u}^h)_n^-$, find $\mathbf{u}^h \in (S_u^h)_n$ and $p^h \in (S_p^h)_n$, such that $\forall\, \mathbf{w}^h \in (V_u^h)_n$ and $\forall\, q^h \in (V_p^h)_n$

$$\int_{Q_n} \mathbf{w}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h\right) dQ + \int_{Q_n} \varepsilon(\mathbf{w}^h) : \sigma(p^h, \mathbf{u}^h)\, dQ$$

$$- \int_{(P_n)_h} \mathbf{w}^h \cdot \boldsymbol{h}\, dP \quad - \int_{(P_n)_{AB}} \mathbf{w}^h \cdot \mathbf{n}_A\, \gamma / R_A\, dP$$

$$+ \int_{Q_n} q^h \rho\, \nabla \cdot \mathbf{u}^h\, dQ + \int_{\Omega} (\mathbf{w}^h)_n^+ \cdot \rho\, ((\mathbf{u}^h)_n^+ - (\mathbf{u}^h)_n^-)\, d\Omega$$

$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau \left[\rho \left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{w}^h\right) - \nabla \cdot \sigma(q^h, \mathbf{w}^h)\right] \cdot$$

$$\left[\rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h\right) - \nabla \cdot \sigma(p^h, \mathbf{u}^h)\right] dQ = 0 \quad , \tag{13}$$

where $(P_n)_{AB}$ is the space-time surface described by the boundary $(\Gamma_t)_{AB}$ as t traverses the time interval (t_n, t_{n+1}).

In the variational formulation given by (13), the following notation is being used:

$$(\mathbf{u}^h)_n^{\pm} = \lim_{\delta \to 0} \mathbf{u}^h(t_n \pm \delta) \ , \tag{14}$$

$$\int_{Q_n} (\ldots)\, dQ = \int_{I_n} \int_{\Omega} (\ldots)\, d\Omega\, dt \ , \tag{15}$$

$$\int_{P_n} (\ldots)\, dP = \int_{I_n} \int_{\Gamma} (\ldots)\, d\Gamma\, dt \ . \tag{16}$$

Remarks

1. If we were in a standard finite element formulation, rather than a space-time one, the Galerkin formulation of (1)-(8) would have consisted of the first five integrals (their spatial versions of course) appearing in equation (13). In the space-time formulation, because the interpolation functions are discontinuous in time, the sixth integral in equation (13) enforces, weakly, the continuity of the velocity in time. The remaining series of integrals in equation [illegible] are the least-squares terms added to the Galerkin variational formulation to assure the numerical stability [illegible] computations. The coefficient τ determines the weight of such added terms.

2. This kind [illegible] stabilization of the Galerkin formulation is referred to as the Galerkin/least-squares (GLS) procedure, and can be considered as a generalization of the stabilization based on the streamline-upwind/Petrov-Galerkin

(SUPG) procedure employed for incompressible flows. It is with such stabilization procedures that it is possible to use elements which have equal-order interpolation functions for velocity and pressure, and which are otherwise unstable.

3. It is important to realize that the stabilizing terms added involve the momentum equation as a factor. Therefore, despite these additional terms, an exact solution is still admissible to the variational formulation given by equation (13).

The coefficient τ used in this formulation is obtained by a simple multi-dimensional generalization of the optimal τ given in Shakib[10] for one-dimensional space-time formulation. The expression for the τ used in this formulation is

$$\tau = \left(\left(\frac{2}{\Delta t}\right)^2 + \left(\frac{2\, \|u^h\|}{h}\right)^2 + \left(\frac{4\nu}{h^2}\right)^2 \right)^{-1/2}, \tag{17}$$

where ν is the kinematic viscosity, and Δt and h are the temporal and spatial "element lengths".

Remarks

4. Because the finite element interpolation functions are discontinuous in time, the fully discrete equations can be solved one space-time slab at a time. Still, the memory needed for the global matrices involved in this method is quite substantial. For example, in two dimensions, the memory needed for space-time formulation (with interpolation functions which are piecewise linear in time) of a problem is approximately four times more compared to using the finite element method only for spatial discretization. However, iteration methods can be employed to substantially reduce the cost involved in solving the linear equation systems arising from the space-time finite element discretization.

5. The kinematical conditions at the interface $(\Gamma_t)_{AB}$ are automatically satisfied because the discretized subdomains $(\Omega_t)_A$ and $(\Omega_t)_B$ share the same nodes at this interface.

6. The additional term (i.e., the fourth integral) in equation (13) enforces the dynamical conditions associated with the interfaces and free-surfaces in the presence of surface tension effects. If the interface is to be interpreted as the free-surface of a single fluid, then the fluid is assumed to occupy subdomain $(\Omega_t)_A$. This variational formulation can of course be easily extended to more than two fluids.

7. For two-liquid flows, the solution and variational function spaces for pressure should include the functions which are discontinuous across the interface.

B. Application to flows involving moving bodies

As a special case of moving bodies let us now consider a freely moving cylinder. The cylinder moves with unknown linear velocity components V_1 and V_2 and angular velocity $\dot{\Theta}$. The temporal evolutions of these additional unknowns depend on the flow field and can be described by writing the Newton's law for the cylinder:

$$\frac{dV_1}{dt} = \frac{D\,(V_1,\ V_2,\ \dot{\Theta},\ U)}{m}, \tag{18}$$

$$\frac{dV_2}{dt} = \frac{L\,(V_1,\ V_2,\ \dot{\Theta},\ U)}{m}, \tag{19}$$

$$\frac{d\dot{\Theta}}{dt} = \frac{T\,(V_1,\ V_2,\ \dot{\Theta},\ U)}{J}, \tag{20}$$

where D, L, and T are the drag, lift and torque on the cylinder, while m and J are its mass and polar moment of inertia. The vector of nodal values of velocity and pressure is denoted by U. Temporal discretization of equations (18)-(20) leads to a set of equations which, in an abstract form, can be written as

$$\mathbf{V} - \mathbf{V}^- = \Delta t \, \mathbf{D} \, (\mathbf{V}^-, \mathbf{V}, \mathbf{U}) \, . \qquad (21)$$

Here **V** (unknown) and $\mathbf{V}^-$ (known) are vectors representing the motion of the cylinder, respectively, inside the current space-time slab and at the end of the previous one. The current slab thickness $t_{n+1} - t_n$ is Δt. For linear-in-time interpolation, equation (21) takes the form

$$\begin{Bmatrix} (V1)^-_{n+1} \\ (V2)^-_{n+1} \\ (\dot{\Theta})^-_{n+1} \\ (V1)^+_n \\ (V2)^+_n \\ (\dot{\Theta})^+_n \end{Bmatrix} - \begin{Bmatrix} (V1)^-_n \\ (V2)^-_n \\ (\dot{\Theta})^-_n \\ (V1)^-_n \\ (V2)^-_n \\ (\dot{\Theta})^-_n \end{Bmatrix} = \Delta t \begin{Bmatrix} \frac{1}{2m}(D^-_n + D^-_{n+1}) \\ \frac{1}{2m}(L^-_n + L^-_{n+1}) \\ \frac{1}{2J}(T^-_n + T^-_{n+1}) \\ \frac{1}{6m}(D^-_n - D^-_{n+1}) \\ \frac{1}{6m}(L^-_n - L^-_{n+1}) \\ \frac{1}{6J}(T^-_n - T^-_{n+1}) \end{Bmatrix} . \qquad (22)$$

Based on the general expression (21), we can write the incremental form of (22) as

$$- \Delta t \left(\frac{\partial \mathbf{D}}{\partial \mathbf{U}}\right) \Delta \mathbf{U} + \left(\mathbf{I} - \Delta t \left(\frac{\partial \mathbf{D}}{\partial \mathbf{V}}\right)\right) \Delta \mathbf{V} = \mathbf{R}_{\mathbf{V}} \, (\mathbf{U}, \mathbf{V}) \, . \qquad (23)$$

Equation (23) is of course coupled with the incremental form of the discrete equation system resulting from (13):

$$(\mathbf{M}^*_{\mathbf{UU}}) \, \Delta \mathbf{U}^{(i)} + (\mathbf{M}^*_{\mathbf{UV}}) \, \Delta \mathbf{V} = \mathbf{R}_{\mathbf{U}} \, (\mathbf{U}, \mathbf{V}) \, . \qquad (24)$$

In computations reported in this article, the system (23)-(24) is solved by a block iteration scheme in which the term $\left(\frac{\partial \mathbf{D}}{\partial \mathbf{V}}\right)$ is neglected. During each iteration, equation (24) is solved for $\Delta\mathbf{U}$ only, using the value of **V** from the previous iteration; and then **V** is updated by (23) while **U** is held constant. However, the full system can, in principle, be solved simultaneously to take advantage of larger time steps afforded by a fully implicit method. Iterating on the solution will still be needed not only because of the nonlinear nature of (1), but also because of the dependence of the element domains Q^e_n on the vector **V**.

Remark

8. In the DSD/ST procedure to facilitate the motion of free-surfaces, interfaces and solid boundaries, we need to move the boundary nodes with the normal component of the velocity at those nodes. Except for this restriction, we have the freedom to move all the nodes any way we would like to. With this freedom, we can move the mesh in such a way that we only need to remesh when it becomes necessary to do so to prevent unacceptable degrees of mesh distortion and potential entanglements. By minimizing the frequency of remeshing we minimize the projection errors expected to be introduced by remeshing. In fact, for some computations, as a byproduct of moving the mesh, we may be able to get a limited degree of automatic mesh refinement, again with minimal projection errors. For example, a mesh moving scheme suitable for a single cylinder drifting in a bounded flow domain is described in Tezduyar et al.[3]. Also by minimizing the frequency of remeshing, we increase the massive parallelization potential of the computations.

IV. The Formulations with the SUPG and PSPG Stabilizations

The space-time formulation, described in the previous section, has the advantage of being able to handle flow problems involving moving boundaries and interfaces, but is quite costly for large-scale problems. The CEBE iteration technique can be used to reduce the cost substantially. For problems that do not involve any moving boundaries and interfaces, one can use even less costly formulations. These formulations are based on finite element discretization in space only, rather than in both space and time. In this section the variational formulations with the SUPG and PSPG stabilization terms are described.

Associated with the finite element discretization of Ω, we define the following finite element interpolation function spaces for the velocity and pressure :

$$S_u^h = \{ u^h \mid u^h \in [H^{1h}(\Omega)]^{n_{sd}}, u^h \doteq g^h \text{ on } \Gamma_g \}, \tag{25}$$

$$V_u^h = \{ w^h \mid w^h \in [H^{1h}(\Omega)]^{n_{sd}}, w^h \doteq 0 \text{ on } \Gamma_g \}, \tag{26}$$

$$S_p^h = V_p^h = \{ q^h \mid q^h \in H^{1h}(\Omega) \}. \tag{27}$$

Here $H^{1h}(\Omega)$ represents the finite-dimensional function space over the spatial domain Ω. This space is formed by using, over the element domains, first-order polynomials in space. The stabilized Galerkin formulation of (1)-(7) can be written as follows: find $u^h \in S_u^h$ and $p^h \in S_p^h$, such that $\forall\, w^h \in V_u^h$ and $\forall\, q^h \in V_p^h$

$$\int_\Omega w^h \cdot \rho \left(\frac{\partial u^h}{\partial t} + u^h \cdot \nabla u^h\right) d\Omega + \int_\Omega \varepsilon(w^h) : \sigma(p^h, u^h)\, d\Omega - \int_{\Gamma_h} w^h \cdot h \, d\Gamma$$

$$+ \int_\Omega q^h \rho \nabla \cdot u^h \, d\Omega$$

$$+ \sum_{e=1}^{n_{el}} \int_{\Omega^e} (\delta^h + \varepsilon^h) \cdot \left[\rho \left(\frac{\partial u^h}{\partial t} + u^h \cdot \nabla u^h\right) - \nabla \cdot \sigma(p^h, u^h)\right] d\Omega = 0 \quad . \tag{28}$$

As it can be seen from equation (28), two stabilizing terms have been added to the standard Galerkin formulation of (1)-(7); the one with δ^h is the SUPG term, and the one with ε^h is the PSPG (pressure-stabilizing/Petrov-Galerkin) term. For definitions of the Petrov-Galerkin functions δ^h and ε^h see Tezduyar et al.[1].

The spatial discretization of equation (28) leads to the following set of non-linear ordinary differential equations:

$$(\mathbf{M} + \mathbf{M}_\delta)\, \mathbf{a} + \mathbf{N}(\mathbf{v}) + \mathbf{N}_\delta(\mathbf{v}) + (\mathbf{K} + \mathbf{K}_\delta)\, \mathbf{v} - (\mathbf{G} + \mathbf{G}_\delta)\, \mathbf{p} = \mathbf{F} + \mathbf{F}_\delta, \tag{29}$$

$$\mathbf{G}^T \mathbf{v} + \mathbf{M}_\varepsilon \mathbf{a} + \mathbf{N}_\varepsilon(\mathbf{v}) + \mathbf{K}_\varepsilon \mathbf{v} + \mathbf{G}_\varepsilon \mathbf{p} = \mathbf{E} + \mathbf{E}_\varepsilon, \tag{30}$$

where v is the vector of unknown nodal values of u^h, **a** is the time derivative of v, and **p** is the vector of nodal values of p^h. The matrices **M**, **N**, **K** and **G** are derived, respectively, from the time-dependent, advective, viscous, and pressure terms. The vector **F** is due to the boundary conditions (5) and (6) (i.e., the g and h terms), whereas the vector **E** is due to the boundary condition (5). The subscripts δ and ε identify the SUPG and PSPG contributions, respectively.

Let us consider the time-integration of equations (29) and (30) by a one-step generalized trapezoidal rule; i.e., given $(u^h)_n$, find $(u^h)_{n+1}$ and $(p^h)_{n+1}$ (this will be referred to as T1 formulation). When written in an incremental form, the T1 formulation leads to

$$\mathbf{M}^* \ \Delta\mathbf{a} - \mathbf{G}^* \ \Delta\mathbf{p} = \mathbf{R} \ , \tag{31}$$

$$(\mathbf{G}^T)^* \ \Delta\mathbf{a} + \mathbf{G}_\varepsilon \ \Delta\mathbf{p} = \mathbf{Q} \ , \tag{32}$$

where

$$\mathbf{R} = \mathbf{F} + \mathbf{F}_\delta - [(\mathbf{M} + \mathbf{M}_\delta)\,\mathbf{a} + \mathbf{N}(\mathbf{v}) + \mathbf{N}_\delta(\mathbf{v})$$
$$+ (\mathbf{K} + \mathbf{K}_\delta)\,\mathbf{v} - (\mathbf{G} + \mathbf{G}_\delta)\,\mathbf{p}] \ , \tag{33}$$

$$\mathbf{Q} = \mathbf{E} + \mathbf{E}_\varepsilon - [\mathbf{G}^T\,\mathbf{v} + \mathbf{M}_\varepsilon\,\mathbf{a} + \mathbf{N}_\varepsilon(\mathbf{v}) + \mathbf{K}_\varepsilon\,\mathbf{v} + \mathbf{G}_\varepsilon\,\mathbf{p}] \ , \tag{34}$$

$$\mathbf{M}^* = \mathbf{M} + \mathbf{M}_\delta + \alpha\,\Delta t\,(\frac{\partial \mathbf{N}}{\partial \mathbf{v}} + \frac{\partial \mathbf{N}_\delta}{\partial \mathbf{v}} + \mathbf{K} + \mathbf{K}_\delta) \ , \tag{35}$$

$$\mathbf{G}^* = \mathbf{G} + \mathbf{G}_\delta \ , \tag{36}$$

$$(\mathbf{G}^T)^* = \mathbf{M}_\varepsilon + \alpha\,\Delta t\,(\frac{\partial \mathbf{N}_\varepsilon}{\partial \mathbf{v}} + \mathbf{K}_\varepsilon + \mathbf{G}^T) \ . \tag{37}$$

The parameter α controls the stability and accuracy of the time integration algorithm.

Remarks

9. The equation systems (31) and (32) can be solved by treating the velocity explicitly in the momentum equation. Since the SUPG and PSPG supplements are applied to all terms in the momentum equation, in explicit computations the coefficient matrix of the pressure equation is generally not symmetric. All explicit T1 computations reported in this paper are based on the symmetrization of the coefficient matrix of the pressure equation, and the results are obtained with 2 passes per time step. In such computations $\mathbf{M}^*$, $\mathbf{G}^*$ and $(\mathbf{G}^T)^*$ are replaced with

$$\mathbf{M}^* = \mathbf{M}_L \ , \tag{38}$$

$$\mathbf{G}^* = \mathbf{G} \ , \tag{39}$$

$$(\mathbf{G}^T)^* = \alpha\,\Delta t\,\mathbf{G}^T \ , \tag{40}$$

where $\mathbf{M}_L$ is the lumped version of the mass matrix $\mathbf{M}$.

10. One can also write a multi-step (T6) time integration formulation for equations (1)-(7). In the T6 formulation the SUPG term is applied only to the sub-steps involving the advective terms. The PSPG term, on the other hand, is applied only to the sub-steps involving the pressure. For details of the T6 formulation and its performance see Tezduyar et al.[1,22].

V. CEBE (Clustered Element-by-Element) Preconditioning

Consider a linear equation system

$$\mathbf{A}\,\mathbf{x} = \mathbf{b} \tag{41}$$

encountered in finite element computation of a problem. Based on the finite element discretization of the problem domain Ω, the matrices $\mathbf{A}$ and $\mathbf{b}$ are formed by adding together their element level constituents; i.e.,

$$\mathbf{A} = \sum_{e=1}^{n_{el}} \mathbf{A}^e \ , \tag{42}$$

$$\mathbf{b} = \sum_{e=1}^{n_{el}} \mathbf{b}^e \ , \tag{43}$$

where n_{el} is the number of elements.

Remarks

11. The domain Ω can also be a space-time domain, in which case the elements are space-time elements.

12. The element level matrices $\mathbf{A}^e$ and $\mathbf{b}^e$ have the same dimensions as the global matrices $\mathbf{A}$ and $\mathbf{b}$, respectively; i.e., $n_{eq} \times n_{eq}$ and $n_{eq} \times 1$, where n_{eq} is the number equations. However, the only non-zero entries for these element level matrices are those corresponding to the nodes of element e, and this fact is taken into account in the implementation.

We assume that direct solution of (41) is not computationally feasible and that we would like to design a good preconditioner to maximize the efficiency of the iterative solution procedure. To achieve this, first we rewrite (41) in a scaled form

$$\tilde{\mathbf{A}}\,\tilde{\mathbf{x}} = \tilde{\mathbf{b}} \ , \tag{44}$$

where

$$\tilde{\mathbf{A}} = \mathbf{W}^{-1/2}\,\mathbf{A}\,\mathbf{W}^{-1/2} \ , \tag{45}$$

$$\tilde{\mathbf{b}} = \mathbf{W}^{-1/2}\,\mathbf{b} \ , \tag{46}$$

$$\tilde{\mathbf{x}} = \mathbf{W}^{1/2}\,\mathbf{x} \ . \tag{47}$$

The scaling matrix $\mathbf{W}$ is defined as

$$\mathbf{W} = \mathit{diag}\,\mathbf{A} \ . \tag{48}$$

Remarks

13. This definition for the scaling matrix **W** is a good one when the matrix **A** is positive-definite. However, when **A** is not positive-definite, the following alternative definition[24] can be used:

$$\mathbf{W} = \mathit{lump}\ \mathbf{M} \tag{49}$$

where *lump* **M** is the lumped version of the mass matrix **M**. It is perhaps reasonable to look into defining a scaling matrix based on a combination of (48) and (49).

14. In scaling a matrix, no matter what the level of that matrix is, the global scaling matrix **W** is the same. For example, the element level matrices $\mathbf{A}^e$ and $\mathbf{b}^e$ are scaled as

$$\tilde{\mathbf{A}}^e = \mathbf{W}^{-1/2}\,\mathbf{A}^e\,\mathbf{W}^{-1/2}\,, \tag{50}$$

$$\tilde{\mathbf{b}}^e = \mathbf{W}^{-1/2}\,\mathbf{b}^e\,. \tag{51}$$

The matrix **A** can be expressed as

$$\mathbf{A} = \mathbf{W} + \sum_{e=1}^{n_{el}} (\mathbf{A}^e - \mathbf{W}^e)\quad . \tag{52}$$

In the scaled form this expression becomes

$$\tilde{\mathbf{A}} = \mathbf{I} + \sum_{e=1}^{n_{el}} \tilde{\mathbf{B}}^e\ , \tag{53}$$

where

$$\tilde{\mathbf{B}}^e = \tilde{\mathbf{A}}^e - \tilde{\mathbf{W}}^e\ , \qquad e = 1, 2, ..., n_{el}\,. \tag{54}$$

The element-by-element (EBE) preconditioning is based on the approximation of (53) by a sequential product of element level matrices. Earlier implementations can be seen in Hughes et al.[23-24]. Various vectorized versions and applications to three-dimensional problems can be found in Hughes and Ferencz[25]. Parallel implementation of the method is achieved in Tezduyar et al.[28] based on the grouped element-by-element (GEBE) approach[27], in which elements are ordered in groups with no inter-element coupling within each group. The number of groups is minimized to minimize the overhead associated with synchronization in parallel computations. Applications in conjunction with the implicit-explicit and adaptive implicit-explicit element grouping can be seen in Tezduyar and Liou[26], Tezduyar et al.[28], Shakib et al.[29] and Liou and Tezduyar[30]. Depending on the form of matrix A, the EBE type preconditioners can be used with the conjugate gradient, GMRES[15], or some other sophisticated search algorithm.

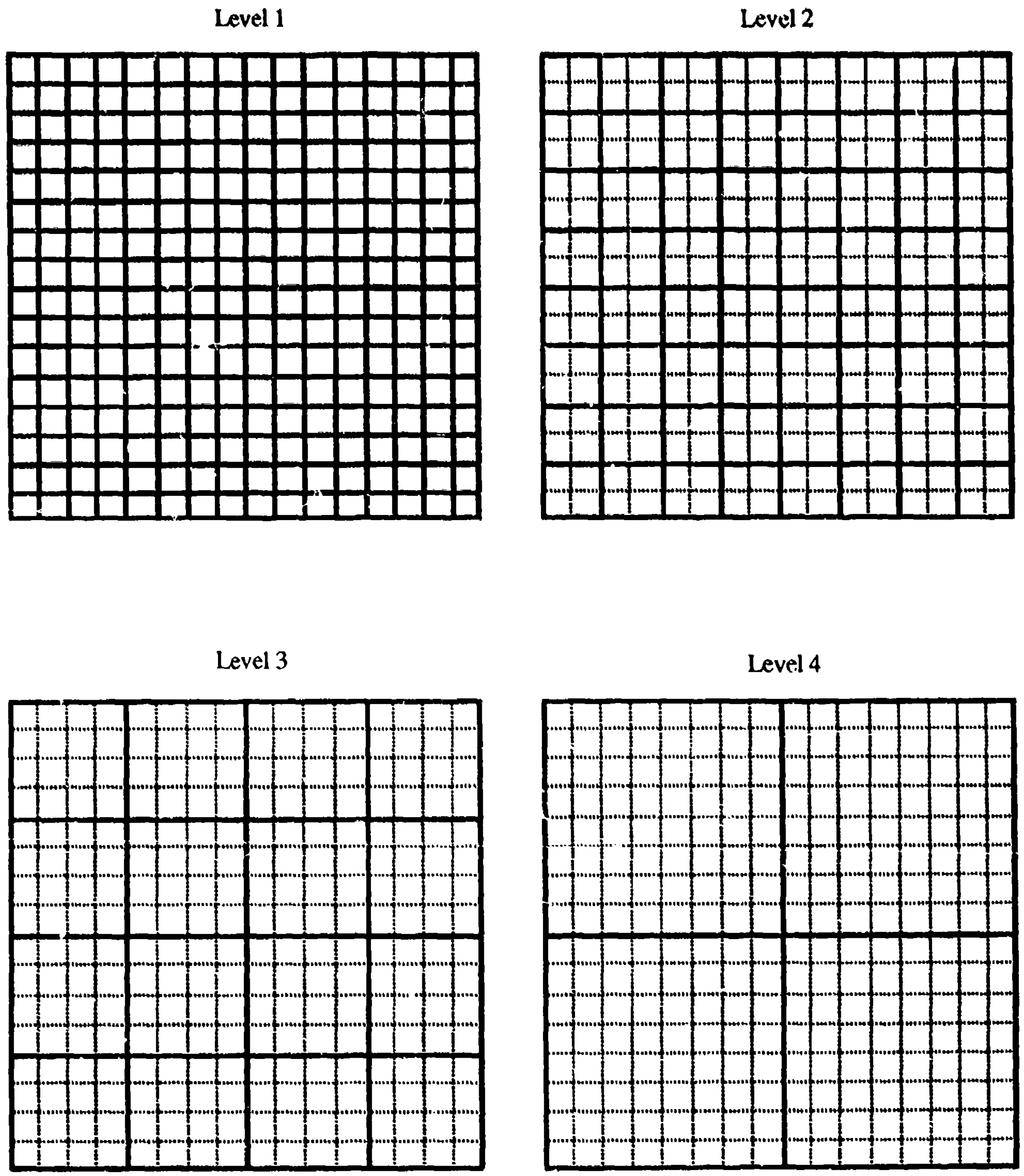


Figure 2. Four different levels of clustering for a uniform 16×16 mesh; in each frame the thick lines depict the cluster boundaries and the associated companion mesh.

In the CEBE (clustered element-by-element) method the set of elements ε is partitioned into clusters of elements ε_J, $J = 1, 2, \ldots, N_{cl}$. For example, Figure 2 shows four different levels of clustering for a uniform 16×16 mesh. The cluster boundaries are marked with thick lines. In the first frame each cluster consists of one element, and therefore this would lead to an EBE method. In the last frame the cluster size is 8×8; the next level of clustering after that (i.e., level 5) would lead to a direct solution method. The global matrix A_J associated with cluster J is defined as

$$A_J = \sum_{e \in \varepsilon_J} A^e \ . \tag{55}$$

The matrix A can then be expressed, similar to (52), as

$$A = W + \sum_{J=1}^{N_{cl}} (A_J - W_J) \ . \tag{56}$$

In the scaled form this expre.   becomes

$$\tilde{A} = I + \sum_{J=1}^{N_{cl}} \tilde{B}_J \ , \tag{57}$$

where

$$\tilde{B}_J = \tilde{A}_J - \tilde{W}_J \ , \qquad J = 1, 2, ..., N_{cl} \ . \tag{58}$$

The CEBE preconditioning is based on the approximation of (57) by a sequential product of cluster level matrices. Here we give two examples (see Liou and Tezduyar[11,31]): 2-Pass CEBE preconditioner and Crout CEBE preconditioner. The 2-Pass CEBE preconditioner is defined as

$$\tilde{P}_C = \prod_{J=1}^{N_{cl}} (I + \frac{1}{2}\tilde{B}_J) \prod_{J=N_{cl}}^{1} (I + \frac{1}{2}\tilde{B}_J) \ , \tag{59}$$

and the Crout CEBE preconditioner is defined as

$$\tilde{P}_C = \prod_{J=1}^{N_{cl}} \hat{L}_J \prod_{J=1}^{N_{cl}} \hat{D}_J \prod_{J=N_{cl}}^{1} \hat{U}_J \ , \tag{60}$$

where $\hat{L}_J$, $\hat{D}_J$ and $\hat{U}_J$ are the matrices resulting from the following Crout factorization:

$$I + \tilde{B}_J = \hat{L}_J \hat{D}_J \hat{U}_J \ , \qquad J = 1, 2, ..., N_{cl} \ . \tag{61}$$

In Liou and Tezduyar[31], these types of preconditioning were used, in conjunction with the conjugate gradient method, for problems governed by the Poisson equation. In Liou and Tezduyar[11] they were used, together with the GMRES method, for compressible and incompressible flow problems. Of course the convergence rates depend on the cluster sizes. In Mittal and Tezduyar[6], for the space-time finite element formulation of an incompressible flow problem, an optimal cluster size was determined by numerical experimentation and was used in the computations.

VI. CC (Cluster Companion) Preconditioning

Let us consider a mesh with different levels of clustering. For each level of clustering in this "primary" mesh, we define a "companion" mesh, such that each cluster of the primary mesh forms an element of the companion mesh. For example, Figure 2 can now also be seen as showing the companion meshes associated with four different levels of clustering in a 16 × 16 primary mesh. In each frame of Figure 2, the thick lines not only mark the cluster boundaries for a certain level of clustering, but also depict the companion mesh associated with that level of clustering. In the first frame the companion mesh is the same as the primary mesh. In the last frame the companion mesh is a 2 × 2 mesh. In our notation, the level of clustering and the associated companion mesh will be identified by the same integer number; i.e., companion mesh l will be associated with clustering level l.

Because the companion mesh 1 is the same as the primary mesh, equations (41)-(43) can also be written as

$$(A)^1 (x)^1 = (b)^1 \tag{62}$$

$$(\mathbf{A})^1 = \sum_{e=1}^{(n_{el})^1} (\mathbf{A}^e)^1 , \tag{63}$$

$$(\mathbf{b})^1 = \sum_{e=1}^{(n_{el})^1} (\mathbf{b}^e)^1 . \tag{64}$$

where the superscript "1" denotes the companion mesh number.

Let $(u)^1$ be the approximation of a displacement-like scalar field u over the companion mesh 1, such that

$$(u)^1 = \sum_{B=1}^{(n_{np})^1} (N)_B^1 (u)_B^1 , \tag{65}$$

where $(n_{np})^1$ is the number of nodal points in companion mesh 1, $(N)_B^1$ is the shape function associated with node B, and $(u)_B^1$ is the value of $(u)^1$ at node B. A similar expression can be written to approximate u over the companion mesh 2:

$$(u)^2 = \sum_{B=1}^{(n_{np})^2} (N)_B^2 (u)_B^2 . \tag{66}$$

Let $(\mathbf{u})^1 = \{ (u)_B^1, B = 1, 2, ..., (n_{np})^1 \}$ and $(\mathbf{u})^2 = \{ (u)_B^2 , B = 1, 2, ..., (n_{np})^2 \}$. Given $(\mathbf{u})^2$, we would like to obtain an expression that approximates $(\mathbf{u})^1$. Based on least-squares minimization of the difference $(u)^1 - (u)^2$, we obtain

$$\mathbf{M}^{11} (\mathbf{u})^1 \cong \mathbf{M}^{12} (\mathbf{u})^2 , \tag{67}$$

where

$$(\mathbf{M}^{11})_{AB} = \int_\Omega (N)_A^1 (N)_B^1 \, d\Omega \quad , \quad A, B = 1, 2, ..., (n_{np})^1 , \tag{68}$$

$$(\mathbf{M}^{12})_{AB} = \int_\Omega (N)_A^1 (N)_B^2 \, d\Omega \, , \quad A = 1, 2, ..., (n_{np})^1 , B = 1, 2, ..., (n_{np})^2 . \tag{69}$$

From (67), we can write

$$(\mathbf{u})^1 \cong \mathbf{E}^{12} (\mathbf{u})^2 , \tag{70}$$

with the "interpolation" matrix $\mathbf{E}^{12}$ defined as

$$\mathbf{E}^{12} = [\mathbf{M}^{11}]^{-1}\,\mathbf{M}^{12} \tag{71}$$

or

$$\mathbf{E}^{12} = [\textit{lump}\ \mathbf{M}^{11}]^{-1}\,\mathbf{M}^{12}\ . \tag{72}$$

An expression similar to (70) can be written to obtain $(\mathbf{u})^2$ from $(\mathbf{u})^1$:

$$(\mathbf{u})^2 \cong \mathbf{E}^{21}\,(\mathbf{u})^1\ , \tag{73}$$

with

$$\mathbf{E}^{21} = [\mathbf{M}^{22}]^{-1}\,\mathbf{M}^{21} \tag{74}$$

or

$$\mathbf{E}^{21} = [\textit{lump}\ \mathbf{M}^{22}]^{-1}\,\mathbf{M}^{21}\ , \tag{75}$$

where

$$(\mathrm{M}^{22})_{AB} = \int_\Omega (\mathrm{N})_A^2\,(\mathrm{N})_B^2\,d\Omega\ ,\quad A, B = 1, 2, \ldots, (n_{np})^2\ , \tag{76}$$

$$(\mathrm{M}^{21})_{AB} = \int_\Omega (\mathrm{N})_A^2\,(\mathrm{N})_B^1\,d\Omega)\ ,\quad A = 1, 2, \ldots, (n_{np})^2\ ,\ B = 1, 2, \ldots, (n_{np})^1\ . \tag{77}$$

More on the derivations related to $\mathbf{E}^{12}$ and $\mathbf{E}^{21}$ can be found in the Appendix. Assuming that the Dirichlet type boundary conditions are somehow taken care of in the implementation, we can also use equations (70) and (73) to obtain $(\mathbf{x})^1$ and $(\mathbf{x})^2$ from each other. That is,

$$(\mathbf{x})^1 \cong \mathbf{E}^{12}\,(\mathbf{x})^2\ , \tag{78}$$

$$(\mathbf{x})^2 \cong \mathbf{E}^{21}\,(\mathbf{x})^1\ . \tag{79}$$

Furthermore, by assuming that $(\mathbf{b})^1$ and $(\mathbf{b})^2$ are force-like quantities and that the energy-like quantities expressed over the two companion meshes, i.e., $(\mathbf{b})^1\,(\mathbf{x})^1$ and $(\mathbf{b})^2\,(\mathbf{x})^2$, are equivalent, we can write

$$(\mathbf{b})^2 \cong \mathbf{F}^{21}\,(\mathbf{b})^1\ , \tag{80}$$

$$(\mathbf{b})^1 \cong \mathbf{F}^{12}\,(\mathbf{b})^2\ , \tag{81}$$

where

$$F^{21} = (E^{12})^T \quad (82)$$

$$F^{12} = (E^{21})^T \quad (83)$$

From (62), (78) and (80) we can write an approximate expression for $[(A)^1]^{-1}$:

$$[(A)^1]^{-1} \cong E^{12}\,[(A)^2]^{-1}\,F^{21} \quad (84)$$

This expression is the starting point for us to construct a companion preconditioner based on the companion mesh 2. The matrix $(A)^2$ can be computed either by using the definition of A over the companion mesh 2, or by using the following expression:

$$(A)^2 \cong F^{21}\,(A)^1\,E^{12} \quad (85)$$

We note, for implementational purposes, that (85) is equivalent to

$$(A)^2 \cong \sum_{c=1}^{(n_{cl})^1} F^{21}\,(A^c)^1\,E^{12} \quad (86)$$

We also note that, if $(A)^1$ is symmetric and positive-definite, so is $(A)^2$ given by the expression (85). However, we cannot say the same thing for $[(A)^1]^{-1}$ given by the expression (84). Therefore, to define our companion preconditioner, we use a regularization similar to the one used in (52). The cluster companion preconditioner based on companion mesh 2 is then defined as

$$[(P_{CC})^{121}]^{-1} = W^{-1} + E^{12}\,(\,[(A)^2]^{-1} - E^{21}\,W^{-1}\,F^{12}\,)\,F^{21} \quad (87)$$

In scaled form, (87) can be rewritten as follows:

$$[(\tilde{P}_{CC})^{121}]^{-1} = I + W^{1/2}\,E^{12}\,(\,[(A)^2]^{-1} - E^{21}\,W^{-1}\,F^{12}\,)\,F^{21}\,W^{1/2} \quad (88)$$

We also experimented with the following modified version of (87):

$$[(P_{CC})^{121}]^{-1} = W^{-1} + E^{12}\,[(A)^2]^{-1}\,F^{21} \quad (89)$$

which can be written in scaled form as

$$[(\tilde{P}_{CC})^{121}]^{-1} = I + W^{1/2}\,E^{12}\,[(A)^2]^{-1}\,F^{21}\,W^{1/2} \quad (90)$$

We can repeat the expressions given by (88) and (90) for the cluster companion preconditioner based on the companion mesh 3:

$$[(\tilde{P}_{CC})^{131}]^{-1} = I + W^{1/2} E^{13} \left([(A)^{3}]^{-1} - E^{31} W^{-1} F^{13} \right) F^{31} W^{1/2} , \tag{91}$$

$$[(\tilde{P}_{CC})^{131}]^{-1} = I + W^{1/2} E^{13} [(A)^{3}]^{-1} F^{31} W^{1/2} . \tag{92}$$

Here E^{13} and E^{31} can be computed either by using definitions similar to those given by equations (71)-(72) and (74)-(75), or by using the following relations:

$$E^{13} = E^{12} E^{23} , \tag{93}$$

$$E^{31} = E^{32} E^{21} . \tag{94}$$

In any case, the matrices F^{31} and F^{13} are defined as

$$F^{31} = (E^{13})^{T} , \tag{95}$$

$$F^{13} = (E^{31})^{T} . \tag{96}$$

Remark

15. It is quite clear that the philosophy behind this type of preconditioning is similar to the philosophy behind multigrid iteration methods.

16. One could also incorporate the idea of "companion" meshes in conjunction with formulations employing higher-order elements. For example, for a mesh using bi-quadratic elements only the nodes at the corners of the higher-order elements will form the companion mesh at level 2. To go to level 3 one could cluster elements in the mesh at level 2 and so on.

VII. Mixed CEBE/CC Preconditioning

It is reasonable to expect that the CEBE preconditioner has more intra-cluster coupling information than the CC preconditioner has. It is also reasonable to expect that the CC preconditioner has more inter-cluster coupling information than the CEBE preconditioner has. Therefore, because these two preconditioners in a sense complement each other, it is reasonable to hope that when they are mixed together they lead to better convergence rates.

Initially our plan was to use these two preconditioners alternately at each iteration of the conjugate gradient method or at each outer iteration of the GMRES method. However, it was recently brought to our attention that Saad[32] has formulated a new version of the GMRES algorithm which allows one to change the preconditioner at every inner iteration. A GMRES subroutine based on this new formulation was made available to us by Saad, and we simply use this subroutine to implement our mixed preconditioning.

In our notation, CEBE-l will represent the CEBE preconditioning based on clustering level l, CC-l will represent the CC preconditioning based on companion mesh l associated with clustering level l, and CEBE/CC-l will represent the mixing of the two. For example, CC-1 would lead to a direct solution method, and therefore we will normally start our test computations with l = 2 or higher.

Remark

17. We note that as l increases, the cost associated with CEBE-l increases and the cost associated with CC-l decreases.

VIII. Numerical Examples for Unsteady Incompressible Flows

All solutions using the space-time formulation, presented here, were obtained with linear-in-time interpolation functions. For the details of the computations see Tezduyar et al.[1-5], Mittal and Tezduyar[6] and Behr et al.[7].

A. *Unsteady flow past a cylinder at Reynolds number 100*

In this test problem the dimensions of the computational domain, normalized by the cylinder diameter, are 30.5 and 16.0 in the flow and cross-flow directions, respectively. The free-stream velocity is 0.125. Reynolds number is based on the free-stream velocity and the diameter of the cylinder. Symmetry conditions are imposed at the upper and lower computational boundaries, and the traction-free condition is imposed at the outflow boundary.

To have a better basis of comparison between the solutions obtained by using Q1Q1 and P1P1 elements, meshes generated with both the elements are required to have the same distribution of the velocity and pressure nodes. The nodal values of the stream function and vorticity are obtained by the least-squares interpolation. For the meshes generated with the P1P1 element, these quantities are computed from the velocity field by using the meshes generated with the Q1Q1 element. For details of the computations and the performance characteristics see Tezduyar et al.[1].

The mesh used for Q1Q1 consists of 5240 elements, while the number of elements for P1P1 is 10,480. Both meshes contain 5350 velocity nodes. A time step of 0.125 was chosen for the computations. The periodic solution is computed by introducing a short term perturbation to the symmetric solution. We have observed, at least for small perturbations, that the periodic solution is independent of the mode of perturbation.

For the Q1Q1/ST computations we use the clustered element-by-element iteration method to solve the resulting equation system. At each time step about 31,500 equations are solved simultaneously. We chose a Krylov vector space of dimension 25 and a cluster size of approximately 25 elements. For this problem, the CEBE technique takes less then one-seventh the CPU time and less then one-third the storage needed by the direct method. It should be mentioned at this point that we have had the successful experience of solving this problem, using the space-time formulation, with a much larger time step (1.0). Here we use a smaller time step to compare the solution with the ones obtained with other formulations.

Strouhal number and the time history of the lift and drag coefficients for the various formulations are shown in Figure 3. The Q1Q1 element gives a Strouhal number about 2% higher than what the P1P1 element gives. Although the lift and drag coefficients show no significant difference among different formulations, the Q1Q1 element gives a slightly higher drag coefficient than the P1P1 element, and the Q1Q1/T1 formulation gives a slightly higher drag coefficient than the Q1Q1/ST formulation.

The periodic solution flow patterns corresponding to the crest value of the lift coefficient are shown in Figures 4-6. The patterns corresponding to the trough value of the lift coefficient are simply the mirror images, with respect to the horizontal centerline, of the patterns shown in Figures 4-6. The solutions obtained with different formulations are very similar. However it can be seen, upon close comparison, that the Q1Q1 element is less dissipative than the P1P1 element and that the Q1Q1/ST formulation shows less dissipation than the Q1Q1/T1 formulation. On comparing these solutions with the ones reported in Tezduyar et al.[33], it can be observed that the solutions obtained with the Q1Q1 and P1P1 elements are very close to the ones obtained with the pQ2P1 and Q1P0/T6. If we compare these solutions with the ones reported for the T6 formulation in Tezduyar et al.[1], we observe that T6 formulation is less dissipative than the T1 formulation and the Q1Q1/ST formulation gives solutions very similar to Q1Q1/T6.

B. *Pulsating drop*

In this problem the drop is initially of elliptical shape with axial dimensions 1.25 (horizontal) and 0.80 (vertical). The density, viscosity and the surface tension coefficient are 1.0, 0.001 and 0.001, respectively. The effect of gravity is dimensions of the drop. Figures 8a, and 8b show the flow field and finite element mesh corresponding, approximately, to points "a", and "b" in Figure 7.

C. *Large-amplitude sloshing*

This problem is similar to the one that was considered in Huerta and Liu[34]. Initially the fluid is stationary and occupies a 2.667 × 1.0 rectangular region. The density and viscosity are 1.0 and 0.002. The gravity is 1.0, and the surface tension is neglected. The wave is created by applying a horizontal body force of A sin(ωt), where A = 0.01 and ω = 0.978. The Reynolds number (based on the height of the fluid and the gravity) is 514. Inviscid boundary conditions are assumed at the walls of the "tank". Compared to the problem considered here, the Reynolds number used in Huerta and Liu[34] is 514,000. Furthermore, in Huerta and Liu[34] the horizontal body force is removed after ten cycles; in this case, on the other hand, this force is maintained during the entire computation. The number of elements is 441, and the time step size is 0.107. With these values of the frequency and the time step size, a single period of the forcing function takes 60 time steps. Figure 9 shows the time history of the vertical location (relative to the stationary level of 1.0) of the free-surface along the left- and right- hand-sides of the "tank". Figures 10a, and 10b show the flow field and finite element mesh corresponding, approximately, to points "a", and "b" in Figure 9.

D. *Flow past an oscillating cylinder*

This is a simple but fundamental fluid-structure interaction problem. The computation involves flow past a circular cylinder that is mounted on flexible supports and is free to respond to the fluid forces in the vertical direction; the Reynolds number for this simulation is 324. The dimensions of the computational domain, normalized by the cylinder radius, are 61.0 and 32.0 in the flow and cross-flow directions, respectively. The mesh employed consists of 4060 elements and 4209 nodes. Symmetry conditions are imposed at the upper and lower computational boundaries, and the traction-free condition is imposed at the outflow boundary. The periodic solution is obtained by introducing a short term perturbation to the symmetric solution. In these computations, we use the CEBE iteration method to solve the resulting equation system. At each time step about 25,000 equations are solved simultaneously. We chose a Krylov vector space of dimension 25 and an average cluster size of 23 elements. For this problem the CEBE technique takes less then one-sixth the CPU time and less then one-third the storage needed by the direct method.

At this Reynolds number the natural frequency of the spring mass system and the vortex shedding frequency for flow past a fixed cylinder have very close values. Consequently, the cylinder undergoes high amplitude oscillations (approximately one cylinder radius) in the vertical direction. These oscillations alter the flow field significantly. Figure 11a shows, for the initial stages of the simulation, time history of the lift, drag and torque coefficients and the normalized vertical displacement and velocity of the cylinder. We observe that the cylinder oscillates with an increasing amplitude. The drag and torque coefficients for the cylinder also increase while the lift coefficient shows a decreasing amplitude. It is interesting to note that both the mean and peak values of the drag coefficient increase with time, but the trough value remains almost constant. The quantities displayed in Figure 11a are shown in Figure 11b for a later stretch of time when the cylinder reaches a steady-state oscillation amplitude of about one radius. The cylinder oscillates with its natural frequency, and so does the torque coefficient; the drag coefficient oscillates with twice the natural frequency of the cylinder. The dominant frequency for the lift coefficient corresponds to the natural frequency of the cylinder. In addition, there is a very small component of the lift coefficient with thrice the frequency of the dominant one. Figure 12 shows a sequence of frames for the vorticity during one period of the cylinder motion. The first, third and last frames correspond to mean cylinder location, while the second and fourth frames correspond, respectively, to the lower and upper extreme positions of the cylinder. For details of this problem and the computations see Mittal and Tezduyar[6].

E. *Flow past an oscillating airfoil*

This computation, performed on the Connection Machine CM-5, involves flow past a NACA 0012 airfoil pitching at Reynolds number 1000. The solution is obtained using the space-time algorithm with Galerkin/least-squares stabilization. The mesh consists of 6609 nodes and 6460 elements. Linear-in-time shape functions are used. At each time step, approximately 39,000 equations are solved simultaneously. The implicit equation system is solved using the GMRES method in conjunction with a diagonal preconditioner. The steady-state solution for flow past a stationary airfoil at an angle of attack of 10 degrees and at Reynolds number 1000 is used as initial condition. Then the airfoil is forced to pitch about its half chord point with a non-dimensional frequency of 1.0 (fc/U). Figure 13 the vorticity field at various instants of the pitching motion. During each period of airfoil oscillation two vortices are shed, one from the leading edge and the other from the trailing edge. For details of this problem and the implementation on the Connection Machine see Mittal and Tezduyar[6] and Behr et al.[7].

elements cover the entire domain and the intersection between elements occurs only on common points, sides or triangular faces in the three dimensional case. The final grid is constructed in a bottom-up manner. The process starts by discretising each boundary curve. Nodes are placed on the boundary curve components and then contiguous nodes are joined with straight line segments. In later stages of the generation process, these segments will become sides of some triangles. The length of these segments must therefore, be consistent with the desired local distribution of grid size. This operation is repeated for each boundary curve in turn.

The next stage consists of generating triangular planar faces. For each two dimensional region or surface to be discretised, all the edges produced when discretising its boundary curves are assembled into the so called initial front. The relative orientation of the curve components with respect to the surface must be taken into account in order to give the correct orientation to the sides in the initial front. The front is a dynamic data structure which changes continuously during the generation process. At any given time, the front contains the set of all the sides which are currently available to form a triangular face. A side is selected from the front and a triangular element is generated. This may involve creating a new node or simply connecting to an existing one. After the triangle has been generated, the front is updated and the generation proceeds until the front is empty. Figure 3.1 illustrates the idea of the advancing front technique for a circular planar domain by showing the initial front and the form of the grid at various stages during the generation process. The size and shape of the generated triangles must be consistent with the local desired size and shape of the final grid. In the three dimensional case, these triangles will become faces of the tetrahedra to be generated later.

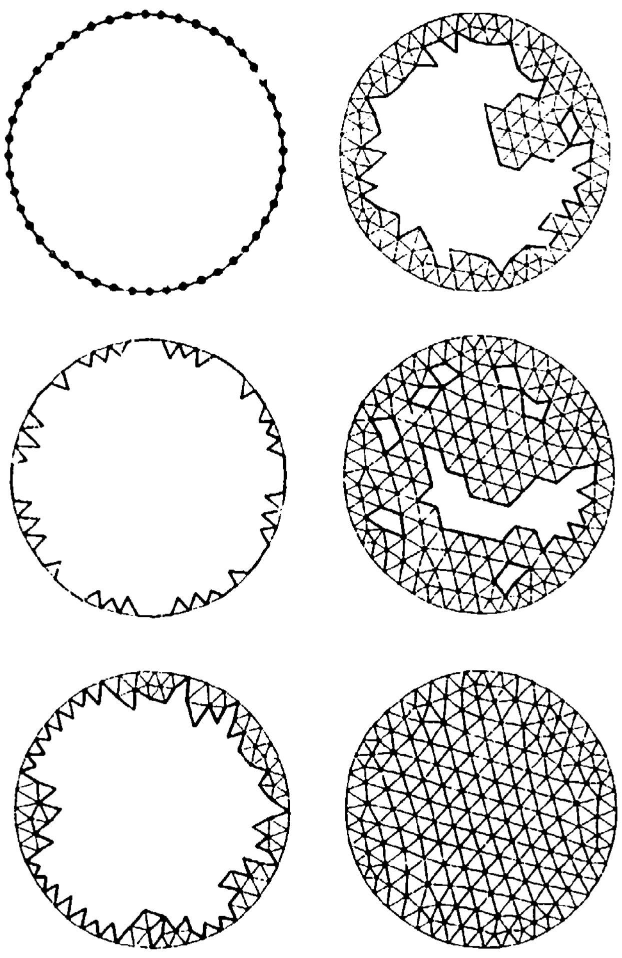


Figure 3.1
The advancing front technique showing different stages during the triangulation process.

For the generation of tetrahedra the advancing front procedure is taken one step further. The front is now made up of the triangular faces which are available to form a tetrahedron. The initial front is obtained by assembling the triangulations of the boundary surfaces. Nodes and elements will be simultaneously created. When forming a new tetrahedron, the three nodes belonging to a triangular face from the front are connected either to an existing node or to a new node. After generating a tetrahedron, the front is updated. The generation procedure is completed when the number of triangles in the front is zero.

3.2 Characterisation of the Grid: Grid Parameters

The geometrical characteristics of a general grid are locally defined in terms of certain *grid parameters*. If N (=2 or 3), is the number of dimensions then, the parameters used are a set of N mutually orthogonal directions $\underline{\alpha}_i$; i=1, ... N, and N associated element sizes δ_i; i=1, ... N (see figure 3.2). Thus, at a certain point, if all N element sizes are equal, the grid in the vicinity of that point will consist of approximately equilateral elements. To aid the grid generation procedure, a transformation T which is a function of $\underline{\alpha}_i$ and δ_i is defined. This transformation is represented by a symmetric N x N matrix and maps the physical space onto a space in which elements, in the neighbourhood of the point being considered, will be approximately equilateral with unit average size. This new space will be referred to as the normalised space. For a general grid this transformation will be a function of position. The transformation T is the result of superimposing N scaling operations with factors $1/\delta_i$ in each $\underline{\alpha}_i$ direction. Thus

$$T(\underline{\alpha}_i, \delta_i) = \sum_{i=1}^{N} \frac{1}{\delta_i} \underline{\alpha}_i \otimes \underline{\alpha}_i \qquad (3.1)$$

where ⊗ denotes the tensor product of two vectors. The effect of this transformation in two dimensions is illustrated in figure 3.3 for the case of constant grid parameters throughout the domain.

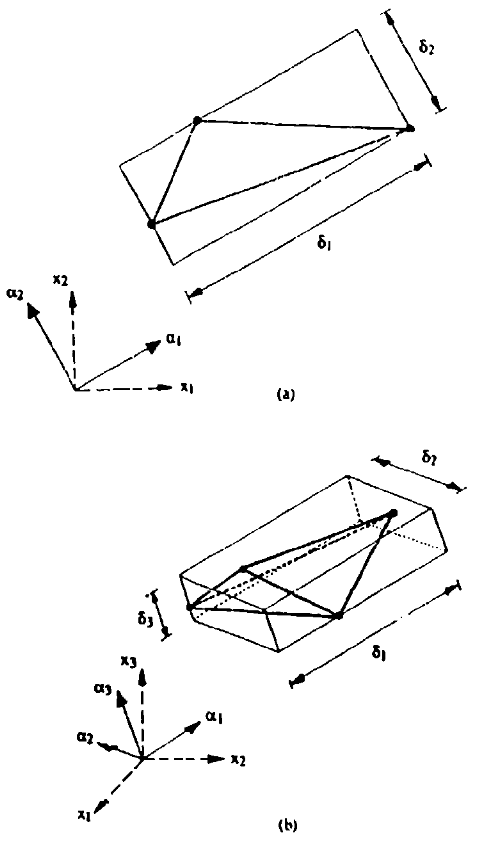


Figure 3.2
Characterisation of the grid. (a) the grid parameters in two dimensions. (b) the grid parameters in three dimensions.

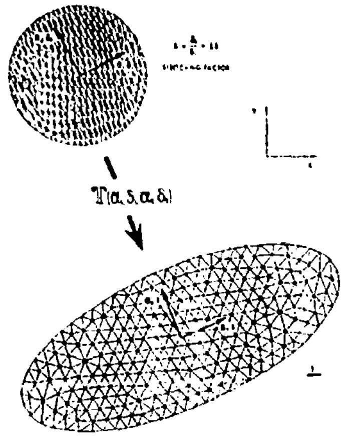


Figure 3.3
The effect of the transformation T for a constant distribution of the grid parameters.

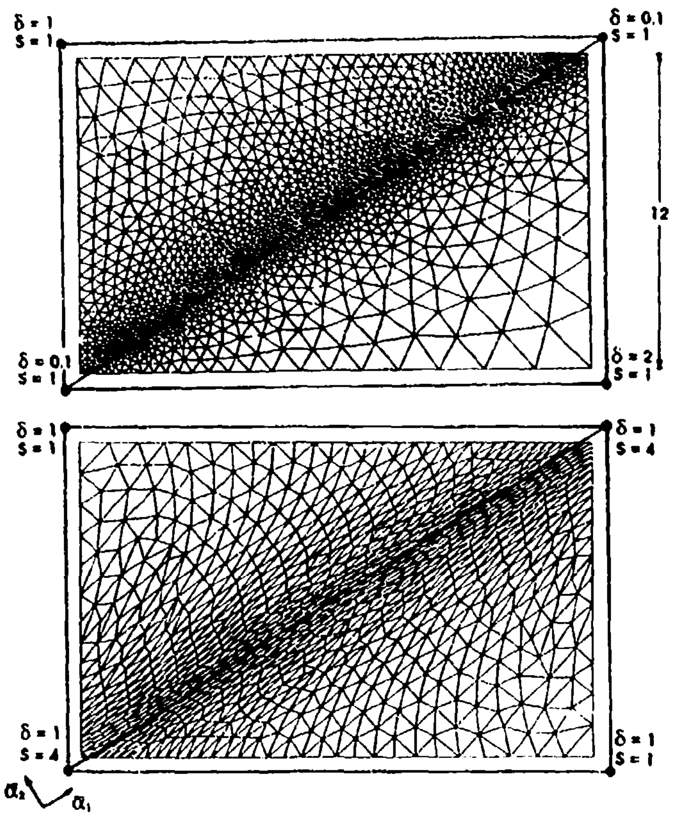


Figure 3.5
Grids generated for a rectangular domain using a background grid consisting of two elements to illustrate the effect of variable grid spacing and stretching.

3.3 Grid Control: The Background Grid

The inclusion of adequate grid control is a key ingredient in ensuring the generation of a grid of the desired form. Control over the characteristics is obtained by the specification of a spatial distribution of grid parameters by means of a *background grid*. The background grid is used for interpolation purposes only and is made up of triangles in two dimensions and tetrahedra in three dimensions. Values of $\underline{\alpha}_i$ and δ_i, and hence T, are defined at the nodes of the background grid. At any point within an element of the background grid, the transformation T is computed by linearly interpolating its components from the element nodal values. The background grid employed must cover the region to be discretised (see figure 3.4). In the generation of an initial grid for the analysis of a particular problem, the background grid will usually consist of a small number of elements. The generation of the background grid can in this case be accomplished without resorting to sophisticated procedures e.g. a background grid consisting of a single element can be used to impose the requirement of linear or constant spacing and stretching through the computational domain. The generation process is always carried out in the normalised space. The transformation T is repeatedly used to transform regions in the physical space into regions in the normalised space. In this way the process is greatly simplified, as the desired size for a side, triangle or tetrahedra in this space is always unity. After the element has been generated, the coordinates of the newly created point, if any, are transformed back to the physical space using the inverse transformation. The effect of prescribing a variable grid spacing and stretching is illustrated in figure 3.5 for a rectangular domain and using a background grid consisting of two triangular elements.

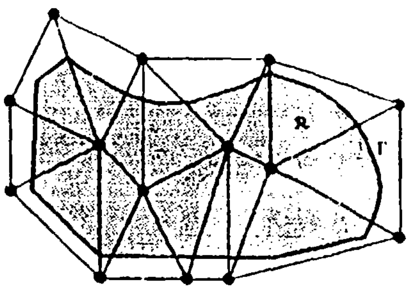


Figure 3.4
The background grid for the specification of a spatial distribution of grid parameters.

3.4 Curve Discretisation

The discretisation of the boundary curve components is achieved by positioning nodes along the curve according to a spacing dictated by the local value of the grid parameters. Consecutive points are joined by straight lines to form sides. In order to determine the position and number of nodes to be created on each curve component, the following steps are followed:

i) Subdivide recursively each cubic segment into smaller cubic segments until their length is smaller than a certain prescribed value. A safe choice for this value is the minimum spacing specified in the background grid but often, considerably larger values can be taken. The length of each cubic segment is computed numerically. When subdividing a cubic segment, the position and tangent vectors corresponding to the new data points can be found directly from the original definition of the segment.

ii) For all the data points $\hat{\underline{r}}_j$; j=1,....,n (i.e. those used to define the curve and those created to satisfy the maximum length criterion), interpolate from the background grid the coefficients of the transformation T_j and transform the position and tangent vectors i.e. $\hat{\underline{r}}_j = T_j \underline{r}_j$ and $\hat{\underline{t}}_j = T_j \underline{t}_j$. The new position and tangent vectors $\hat{\underline{r}}_j$, $\hat{\underline{t}}_j$; j=1,...,n, define a spline curve which can be interpreted as the image of the original curve component in the normalised space. It must be noted that because of the approximate nature of this procedure, the new curve will in general have discontinuities of curvature even though the curvature of the original curve varies continuously.

iii) Compute the length of the curve in the normalised space and subdivide it into segments of approximately unit length. For each newly created point, calculate the cubic segment in which it is contained and its parametric coordinate. This information is used to determine the coordinates of the new nodes in the physical space, using the curve component definition.

3.5 Triangle Generation in Two Dimensional Domains
The triangle generation algorithm utilises the concept of a generation front. At the start of the process the front consists of the sequence of straight line segments which connect consecutive boundary nodes. During the generation process, any straight line segment which is available to form an element side is termed active, whereas any segment which is no longer active is removed from the front. Thus while the domain boundary will remain unchanged, the generation front changes continuously and needs to be updated whenever a new element is formed. This updating process is illustrated in figure 3.6.

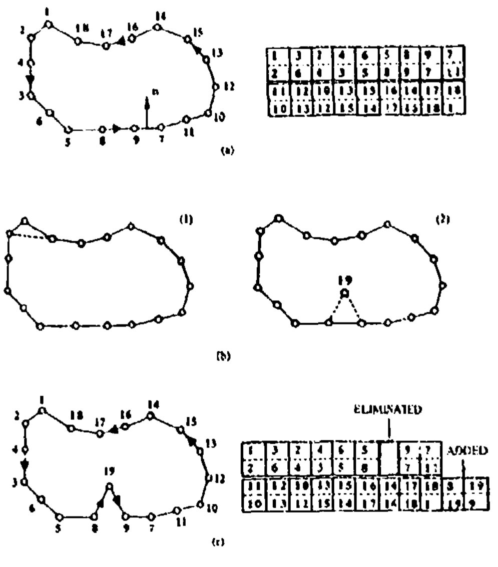


Figure 3.6
The front updating procedure in two dimensions. (a) The initial generation front. (b) Creation of a new element with (1) no new point created (2) the new point 19 is created. (c) The updating of the front for case (b)(2).

In the process of generating a new triangle the following steps are involved (figure 3.7):

i) Select a side AB of the front to be used as a base for the triangle to be generated. Here, the criterion is to choose the shortest side. This is especially advantageous when generating irregular grids.
ii) Interpolate from the background grid the transformation **T** at the centre of the side $\underline{M}$ and apply it to the nodes in the front which are relevant to the triangulation. In our implementation we define the relevant points to be all those which lie inside the circle of centre $\underline{M}$ and radius three times the length of the side being considered. Let $\underline{\hat{A}}$, $\underline{\hat{B}}$ and $\underline{\hat{M}}$ denote the positions in the normalised space of the points $\underline{A}$, $\underline{B}$ and $\underline{M}$ respectively.
iii) Determine, in the normalised space, the ideal position $\underline{\hat{P}}_1$ for the vertex of the triangular element. The point $\underline{\hat{P}}_1$ is located on the line perpendicular to the side that passes through the point $\underline{M}$ and at a distance δ_1 from the points $\underline{\hat{A}}$ and $\underline{\hat{B}}$. The direction in which $\underline{\hat{P}}_1$ is generated is determined by the orientation of the side. The value δ_1 is chosen according to:

$$\delta_1 = \begin{cases} 1 & \text{if} \quad 0.55 \cdot L < 1 < 2 \cdot L \\ 0.55 \cdot L & \text{if} \quad 0.55 \cdot L < 1 \\ 2 \cdot L & \text{if} \quad 1 > 2 \cdot L \end{cases} \tag{3.2}$$

where L is the distance between points $\underline{\hat{A}}$ and $\underline{\hat{B}}$. Only in situations where the side AB happens to have characteristics very different from those specified by the background grid will the value of δ_1 be different from unity. However, the above inequalities must be taken into account to ensure geometrical compatibility. Expression (3.2) is purely empirical and different inequalities could be devised to serve the same purpose.

iv) Select other possible candidates f[illegible] vertex and order them in a list. Two types of points are considered viz. (a) all the nodes $\underline{\hat{Q}}_1$, $\underline{\hat{Q}}_2$ ... in the current generation front which are, in the normalised space, interior to a circle with centre $\underline{\hat{P}}_1$ and radius $r = \delta_1$, and (b) the set of points $\underline{\hat{P}}_1, \ldots, \underline{\hat{P}}_5$ generated along the height $\underline{\hat{P}}_1\underline{\hat{M}}$. For each point $\underline{\hat{Q}}_i$, construct the circle with centre $\underline{\hat{Q}}_i$ on the line defined by points $\underline{\hat{P}}_1$ and $\underline{\hat{M}}$ and which passes through the points $\underline{\hat{Q}}_i$, $\underline{\hat{A}}$ and $\underline{\hat{B}}$. The position of the centres $\underline{\hat{Q}}_i$ of these circles on the line $\underline{\hat{P}}_1\underline{\hat{M}}$ defines an ordering of the the $\underline{\hat{Q}}_i$ points. A list is created which contains all the $\hat{Q}$ points with the furthest point from $\underline{\hat{P}}_1$ appearing at the head of list. The points $\underline{\hat{P}}_1, \ldots, \underline{\hat{P}}_5$ are added at the end of this list.
v) Select the best connecting point. This is the first point in the ordered list which gives a consistent triangle. Consistency is guaranteed by ensuring that none of the newly created sides intersects with any of the existing sides in the front.
vi) Finally, if a new node is created, its coordinates in the physical space are obtained by using the inverse transformation T^{-1}.
vii) Store the new triangle and update the front by adding/removing the relevant sides.

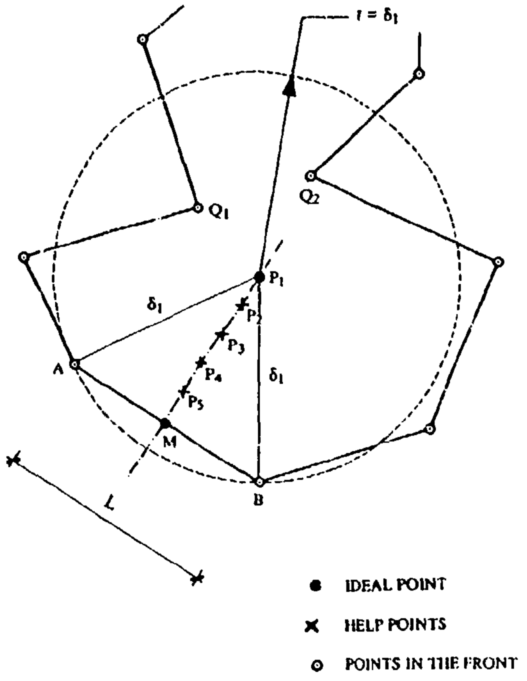


Figure 3.7
The generation of a new triangle.

This grid generation procedure is schematically presented in the diagram shown in figure 3.8

Grid Quality Enhancement
In order to enhance the quality of the generated grid, two post-processing procedures are applied. These procedures, which are local in nature, do not alter the total number of points or elements in the grid.
Diagonal swapping - this changes the connectivities among nodes in the grid without altering their position. This process requires a loop over all the element sides excluding those sides on the boundary. For each side AB (figure 3.9) common to the triangles ABC and ADB one considers the possibility of swapping AB by CD, thus replacing the two triangles ABC and ADB by the triangles ADC and BCD. The swapping is performed if a prescribed regularity criterion is satisfied better by the new configuration than by the existing one. In our implementation, the swapping operation is performed if the minimum angle occuring in the new configuration is larger than in the original one.

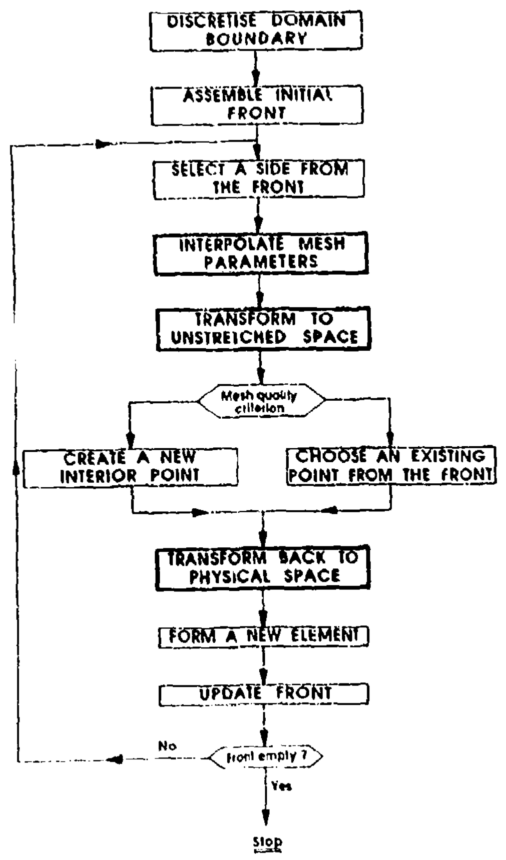

DISCRETISE DOMAIN BOUNDARY
ASSEMBLE INITIAL FRONT
SELECT A SIDE FROM THE FRONT
INTERPOLATE MESH PARAMETERS
TRANSFORM TO UNSTRETCHED SPACE
Mesh quality criterion
CREATE A NEW INTERIOR POINT
CHOOSE AN EXISTING POINT FROM THE FRONT
TRANSFORM BACK TO PHYSICAL SPACE
FORM A NEW ELEMENT
UPDATE FRONT
No
front empty ?
Yes
Stop

Figure 3.8
Grid generation using the advancing front technique. Double lined boxes are only required if the effects of variable grid size and stretching are to be included.

Grid smoothing - this alters the positions of the interior nodes without changing the topology of the grid. The element sides are considered as springs of stiffness proportional to the length of the side. The nodes are moved until the spring system is in equilibrium. The equilibrium positions are found by iteration. Each iteration amounts to performing a loop over the interior points and moving their coordinates to coincide with those of the centroid of the neighbouring points. Usually three to five iterations are performed.

The combined application of these two post-processing algorithms is found to be very effective in improving the smoothness and regularity of the generated grids.

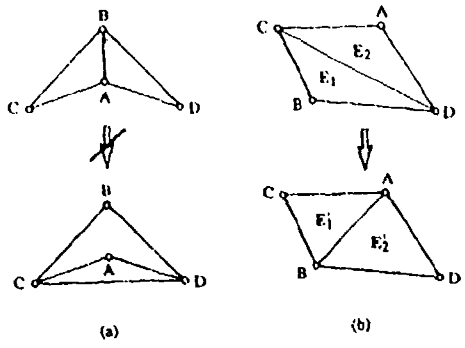


Figure 3.9
The diagonal swapping procedure. (a) non-admissible. (b) admissible.

3.6 Surface Discretisation.
The method followed for the triangulation of the surface components is an extension of the grid generation procedure for planar domains described above. The discretisation of each surface component is accomplished by generating a two dimensional grid of triangles in the parametric plane (u^1,u^2) and then using the mapping $\underline{r}(u^1,u^2)$ defined in section 2.2. This mapping establishes a one to one correspondence between the boundary surface component and a region on the parametric plane (u^1,u^2) (figure 3.10). Thus, a consistent triangular grid in the parametric plane will be transformed, by the mapping $\underline{r}(u^1,u^2)$, into a valid triangulation of the surface component. The construction of the triangular grid in the parameter plane (u^1,u^2) using the two dimensional grid generator, requires the determination of an appropriate spatial distribution of the two dimensional grid parameters. These consist of a set of two mutually orthogonal directions $\underline{\alpha}_i^*$; i=1, 2, and two associated element sizes δ_i^*; i=1, 2.

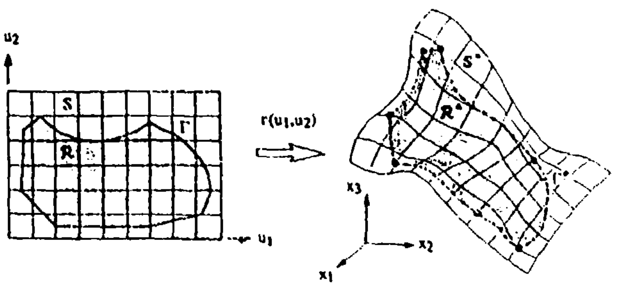


Figure 3.10
The mapping of a surface component onto a two dimensional domain.

The two dimensional grid parameters in the (u^1,u^2) plane can be evaluated from the spatial distribution of the three dimensional grid parameters and the distortion and stretching introduced by the mapping. To illustrate this process, consider a point $\underline{P}^*$ in the parametric plane of coordinates (u^{1P},u^{2P}) where the values of the grid parameters $\delta_i^*, \underline{\alpha}_i^*$; i=1,2 are to be computed. Its image on the surface will be the point $\underline{P} = \underline{r}(u^{1P},u^{2P})$. The transformation between the physical space and the normalised space at this point T_P can be obtained by direct interpolation from the background grid. A new mapping, valid in the neighbourhood of point P, can now be defined between the parametric plane (u^1,u^2) and the normalised space as

$$\underline{R}(u^1,u^2) = T_P\,\underline{r}(u^1,u^2) \qquad (3.3)$$

A curve in the parametric plane passing through point $\underline{P}^*$ and with unit tangent vector $\underline{\beta} = (\beta^1,\beta^2)$ at this point, is transformed by the above mapping into a curve in the normalised space passing through the point $T_P\,\underline{P}$. The arc length parameters ds and $d\zeta$, along the original and transformed curves respectively, are related by the expression [35]

$$(d\zeta)^2 = \left\{ \sum_{i,j=1}^{2} \frac{\partial \underline{R}}{\partial u^i}\frac{\partial \underline{R}}{\partial u^j}\,\beta^i\,\beta^j \right\}(ds)^2 \qquad (3.4)$$

Assuming that this relation between the arc length parameters also holds for the spacings, we can compute the spacing δ_β along the direction $\underline{\beta}$ in the parameter plane as

$$\delta_\beta = \left\{ \sum_{i,j=1}^{2} \frac{\partial \underline{R}}{\partial u^i}\frac{\partial \underline{R}}{\partial u^j}\,\beta^i\,\beta^j \right\}^{1/2} \qquad (3.5)$$

The two dimensional grid parameters $\underline{\alpha}_i^*, \delta_i^*$; i=1, 2 are determined from the directions in which δ_β attains an extremum. This reduces to finding the eigenvalues and eigenvectors of a symmetric 2 x 2 matrix.

To form the initial front, the (u^1,u^2) coordinates of the nodes already generated on the boundary curve components have to be computed. As the mapping $\underline{r}(u^1,u^2)$ cannot be inverted analytically, the coordinates (u^1,u^2) of such points are found numerically by using a direct iteration procedure [31].

3.7 Generation of Tetrahedra

The starting point for the discretisation of the three dimensional domain into tetrahedra is the formation of an initial generation front. The initial front is the set of oriented triangles which constitutes the discretised boundary of the domain and is formed by assembling the discretised boundary surface components. The order in which the nodes of these triangles are given defines the orientation, which is the same as that of the corresponding boundary surface component. The algorithm for generating tetrahedra is analogous to that described above for the generation of triangles (see figure 3.8). However, in the three dimensional case the range of possible options at each stage is much wider and the number of geometrical operations involved increases considerably. Thus, the ability of the method to produce a grid and the efficiency of its implementation relies heavily upon the type of strategy selected. The generation of a generic tetrahedral element involves the following steps (figure 3.11):

i) Select a triangular face ABC from the front to be a base for the tetrahedron to be generated. In principle, any face could be chosen, but we have found it to be advantageous in practice to consider the smallest faces first. For this purpose, the size of the face is defined in terms of the size of its shortest height.

ii) Interpolate from the background grid the transformation T at the centroid of the face $\underline{M}$ and apply it to the nodes in the front which are relevant to the triangulation. In our implementation, we define the relevant points to be those which lie inside the sphere of centre $\underline{M}$ with radius equal to three times the value of the maximum dimension of the face being considered. Let $\hat{\underline{A}}$, $\hat{\underline{B}}$, $\hat{\underline{C}}$ and $\hat{\underline{M}}$ denote the positions in the normalised space of the points $\underline{A}$, $\underline{B}$, $\underline{C}$ and $\underline{M}$ respectively.

iii) Determine, in the transformed space, the ideal position $\hat{\underline{P}}_1$ for the vertex of the tetrahedral element. The point $\hat{\underline{P}}_1$ lies on the line which passes through the point $\hat{\underline{M}}$ and which is perpendicular to the face. The direction in which $\hat{\underline{P}}_1$ is generated is determined by the orientation of the face. The location of $\hat{\underline{P}}_1$ is computed so that the average length of the three newly created sides which join point $\hat{\underline{P}}_1$ with points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$ is unity. For faces whose size in the parametric plane is very different from unity, this step may have to be modified, as in expression (3.2), to ensure geometrical compatibility. However, such cases rarely occur in practice. Let δ_1 be the maximum of the distances between point $\hat{\underline{P}}_1$ and points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$.

iv) Select other possible candidates for the vertex and order them in a list. Two types of points are considered viz. (a) all the nodes $\hat{\underline{Q}}_1$, $\hat{\underline{Q}}_2$, ... in the current generation front which are, in the normalised space, interior to a sphere with centre $\hat{\underline{M}}$ and radius $r = \delta_1$, and (b) a new set of points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ generated along the height $\hat{\underline{P}}_1\hat{\underline{M}}$. Consider the set of points $\hat{\underline{A}}$, $\hat{\underline{B}}$ and $\hat{\underline{C}}$ and denote by $\hat{\underline{D}}$ the member of this set which is furthest away from $\hat{\underline{M}}$. For each point $\hat{\underline{Q}}_i$, construct the sphere with centre $\hat{\underline{O}}_i$ on the line defined by points $\hat{\underline{P}}_1$ and $\hat{\underline{M}}$ and which passes through point $\hat{\underline{Q}}_i$ and $\hat{\underline{D}}$. The position of the centres $\hat{\underline{O}}_i$ of these spheres on the line $\hat{\underline{P}}_1\hat{\underline{M}}$ defines an ordering of the the $\hat{\underline{Q}}_i$ points with the furthest point from $\hat{\underline{P}}_1$ appearing at the head of list. The points $\hat{\underline{P}}_1, \ldots, \hat{\underline{P}}_5$ are added at the end of this list.

v) Select the best connecting point. This is the first point in the ordered list which gives a consistent tetrahedron. Consistency is guaranteed by ensuring that none of the newly created sides intersects with any of the existing faces in the front, and that none of the existing sides in the front intersect with any of the newly created faces.

vi) If a new node is created, its coordinates in the physical space are obtained by using the inverse transformation T^{-1}.

vii) Store the new triangle and update the front by adding/removing the necessary sides.

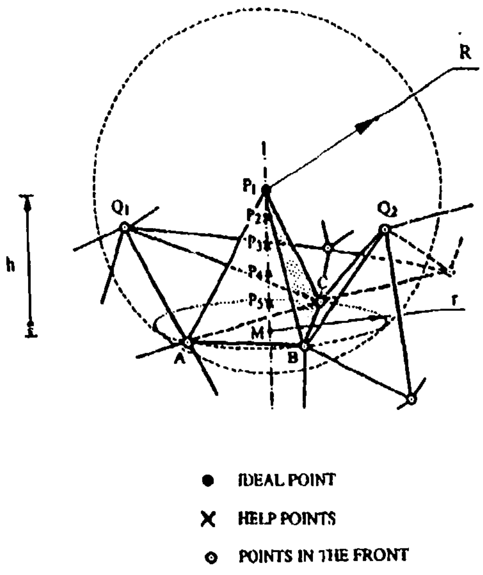


Figure 3.11
The generation of a tetrahedral element.

3.8 Grid Quality Assessment

Any discussion of grid quality should be intimately related to the form of the solution we are trying to represent on that grid. Two factors need to be considered here:

i) Determination of the characteristics of the optimal grid for the problem at hand. This introduces the concept of adaptivity and this aspect is considered in section 5.

ii) Assessment on how well the generated grid meets the requirements specified by the grid parameters. This assessment can be made by examining the generated grid and determining the statistical distribution of certain indicators. For example in figure 3.12 we have chosen as indicators the number of elements around a side, the magnitude of the element dihedral angles and the length of the side. These indicators are compared with optimal values i.e. those of a regular tetrahedron which has the exact dimensions specified by the grid parameters.

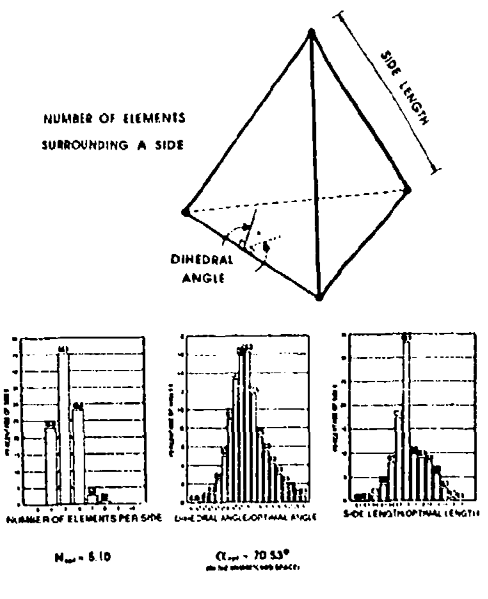
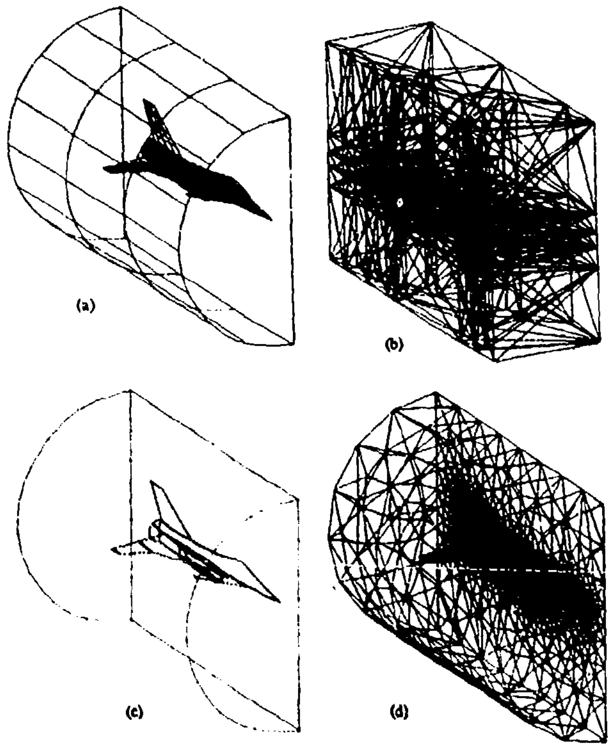


Figure 3.12
Grid quality statistics.

3.9 Grid Generation for a Generic Fighter Configuration

In computational aerodynamics, a problem of current interest is the prediction of the inviscid flowfield about complete aircraft configurations. The problem considered here is the simulation of the flow past a generic fighter with canard, 70-20 cranked delta wing, vertical fin and engine inlet. This same configuration has been studied previously using an algebraic grid generation approach [36]. Due to the symmetry of the problem only half of the fighter is modelled. Figure 3.13(a) shows the geometry definition of the computational domain. The background grid employed is illustrated in figure 3.13(b). The curve components, defined in terms of cubic splines and the discretisation of these components is displayed in figure 3.13(c). The individual surface components are described by tensor product surfaces and the surface discretisation is illustrated in figure 3.13(d). An intermediate stage during the tetrahedra generation process is displayed in figure 3.13(e). The final grid consisted of 76,522 tetrahedra and included a full simulation of the



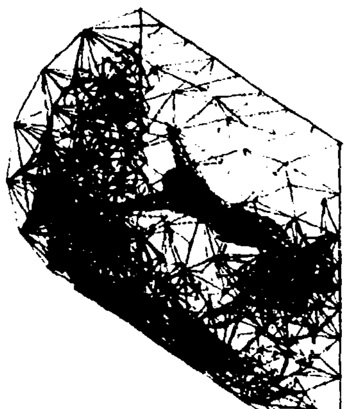
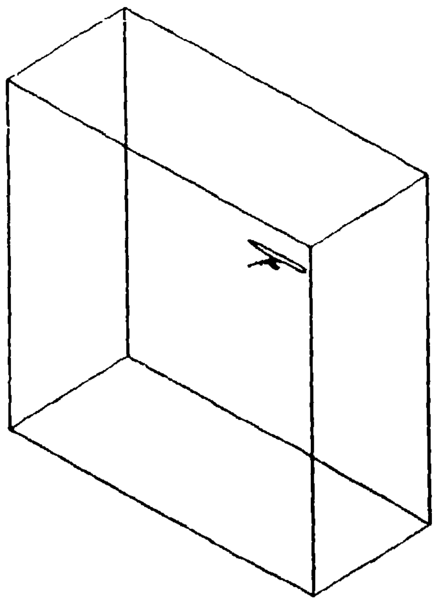


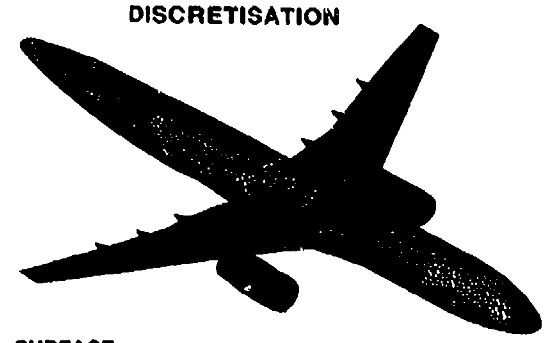
Figure 3.13
Grid generation for a complete aircraft configuration. (a) the computational domain and the geometry definition. (b) the background grid (c) the representation of the curve components and the generated points. (d) the surface discretisation. (e) A partial view of the tetrahedral grid.

3.10 The Computation of Transonic Flow Past an Installed Nacelle

To illustrate the numerical performance of the unstructured mesh solution procedure, we consider the analysis of a transonic inviscid flow past a realistic wing/pylon/nacelle configuration [69]. The configuration has been tested in a wind tunnel with a turbine powered simulator to represent the flow into the engine and the jet emerging from the exhaust. The assumed symmetry of the problem means that only one half of the configuration needs to be considered. The definition of the computational domain is completed by the addition of the symmetry plane and planar far field boundaries, as shown in figure 3.14a. Following the triangulation of the computational boundaries, the aircraft surface is represented by an assembly of 36,330 triangles with 19,167 nodal points. Two different views of the surface triangulation are given in figure 3.14b. Starting from this collection of triangular faces, the computational domain is filled with tetrahedra by the advancing front method, simultaneously creating interior nodal points and elements. The 3D grid which is generated contains 592,380 tetrahedra and 112,198 nodes. For the simulation, a free stream Mach number of 0.801 and an angle of attack of 2.738 degrees are assumed. The flow within the engine is simulated by computing a pressure ratio, engine mass flow rate and a jet total temperature by assuming that the separate core and fan streams in the experiment mix together completely within the turbine powered simulator, before exhausting through the nozzle. Starting from free stream conditions, the solution is advanced for 2,500 time-steps, during which time the residual is reduced by four orders of magnitude. The computed distribution of the pressure contours on the surface of the model is shown in figure 3.14c. In the experimental configuration, pressure guages are distributed at fixed sections over the surface of the wing and also on the surface of the nacelle. A comparison of the computed and the experimentally observed values of the pressure coefficient at a section along the wing and on the surface of the nacelle is given in figure 3.14d. The agreement between the numerically predicted values and the experimentally determined pressure data is very good, apart from the over prediction of pressures on the upper surface of the wing. The discrepancies between the experimentally observed pressure distributions and the predictions of the Euler flow solver on the wing upper surface exist in regions where significant viscous effects can be expected to occur.

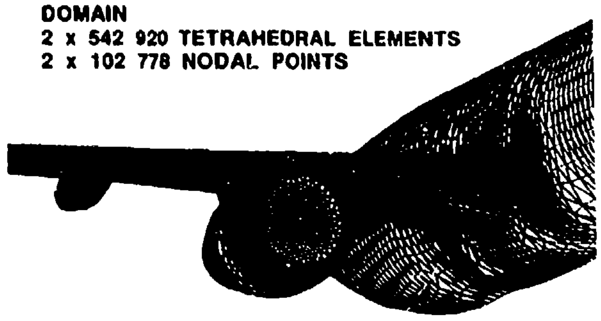

(a)

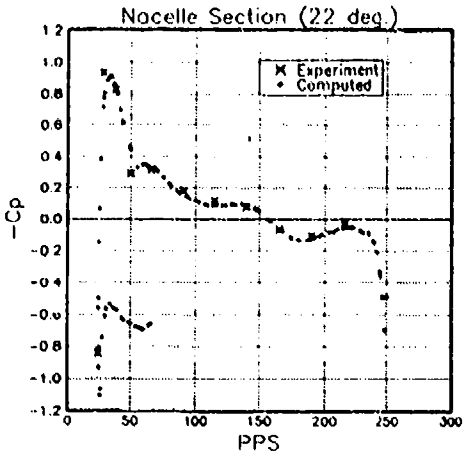
DISCRETISATION



SURFACE
2 x 35 530 TRIANGULAR FACES
DOMAIN
2 x 542 920 TETRAHEDRAL ELEMENTS
2 x 102 778 NODAL POINTS

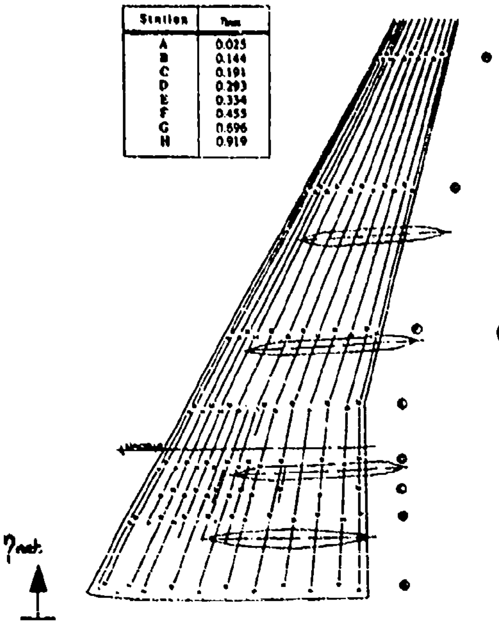
(b)





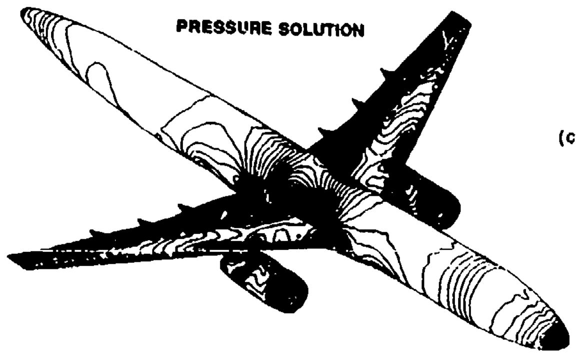
(d)

Station	η_{ref}
A	0.025
B	0.144
C	0.191
D	0.293
E	0.334
F	0.455
G	0.696
H	0.919

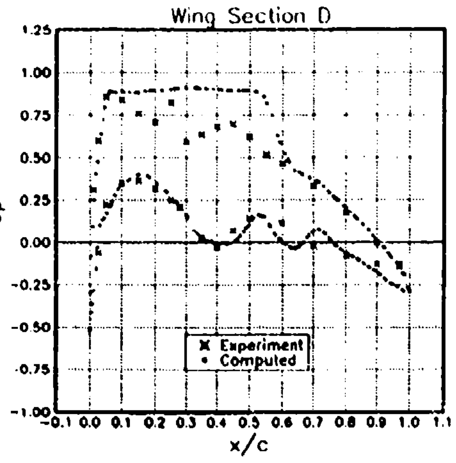


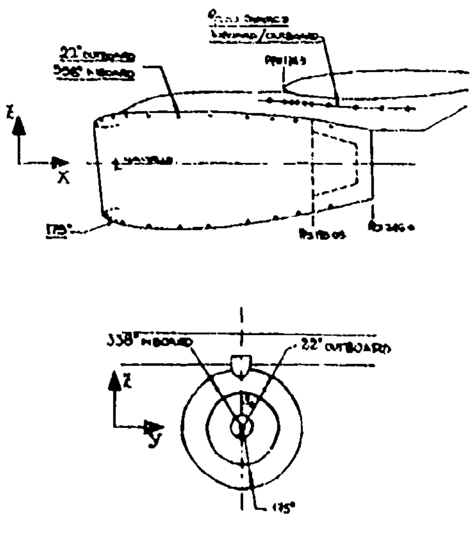
(d)

PRESSURE SOLUTION



(c)





(d)

(d)

Figure 3.14
Simulation of flow over a wing/body/pylon/nacelle configuration at a free stream Mach number of 0.801 and at 2.738 degrees angle of attack. (a) Wireframe of the computational domain. (b) Two views of the surface triangulation. (c) The computed pressure contour distribution. (d) The comparison between the computed and the experimentally measured pressure coefficient at a wing and at a nacelle section.

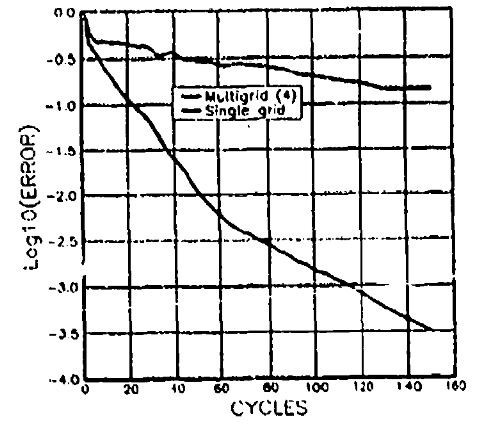
3.11 The Use of Multi-Grid Acceleration

The sequence of unstructured grids required for a multi-grid algorithm of the type described above is readily produced by the grid generator by simply altering the scaling of the user specified distribution functions.

Transonic Flow in a Channel

The multi-grid scheme is applied first to the solution of transonic flow in a channel. This is a 3D simulation of a 2D flow past a 4% thick circular bump. The free stream conditions correspond to a Mach number of 0.675. The computation is performed by using a fine grid of 28,822 tetrahedra and three coarser grids of 4,137, 556 and 119 tetrahedra respectively. The corresponding trianguiations of the boundary of the computational domain are shown in figures 3.15a to 3.15d. Figure 3.15e displays the pressure solution after 150 multi-grid V-cycles (*icycle* =1) with *n1* =1, *n2* =1 and *n3* =0. The increase in the rate of convergence towards the steady state by the use of the multi-grid scheme is readily observed in figure 3.15f which compares the convergence history of the flow calculation on the finest grid with the one obtained using the multi-grid scheme with four grids.

(f)



(a)

(b)

(c)

(d)

(e)

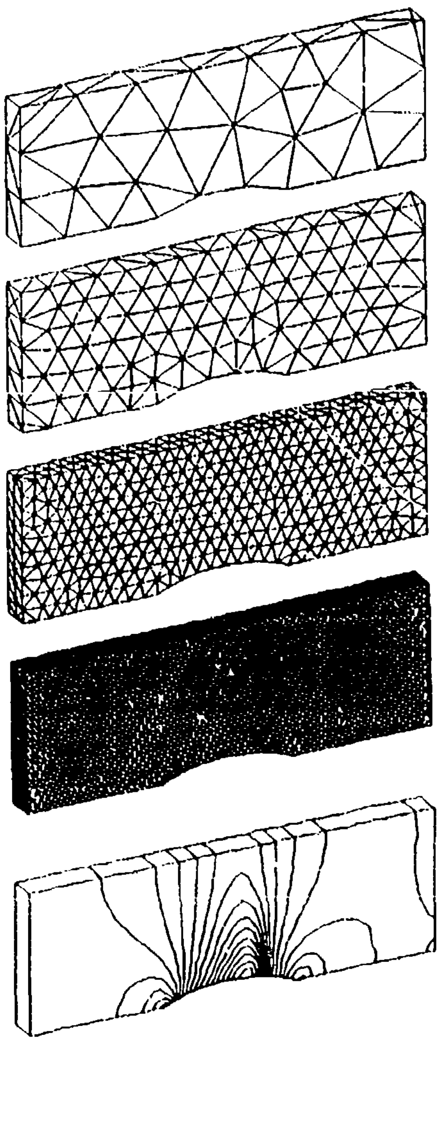
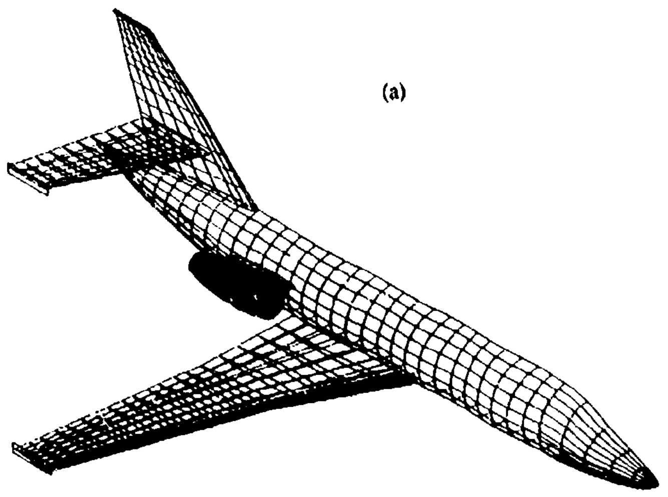


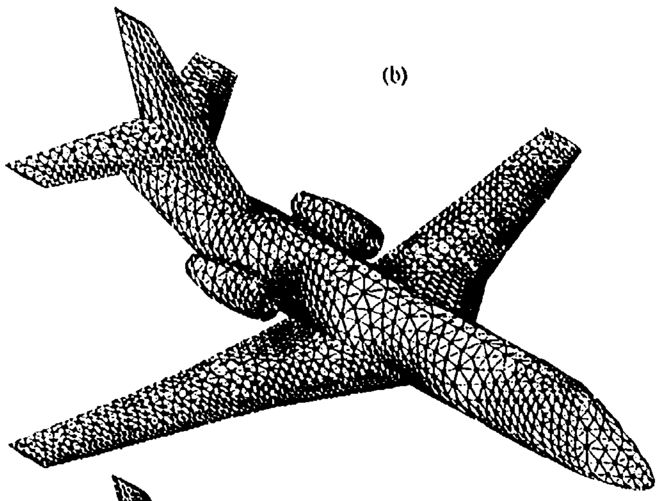
Figure 3.15
Transonic flow in a channel at a free stream Mach number of 0.675 (a) First grid of 119 elements, 63 points. (b Second grid of 565 elements, 216 points. (c) Third grid of 4,137 elements, 1,142 points. (d) Fourth grid of 28,822 elements , 6,709 points. (e) Computed pressure contours. (f) Comparison of the convergence histories produced by using the fine grid alone and by using multi-grid.

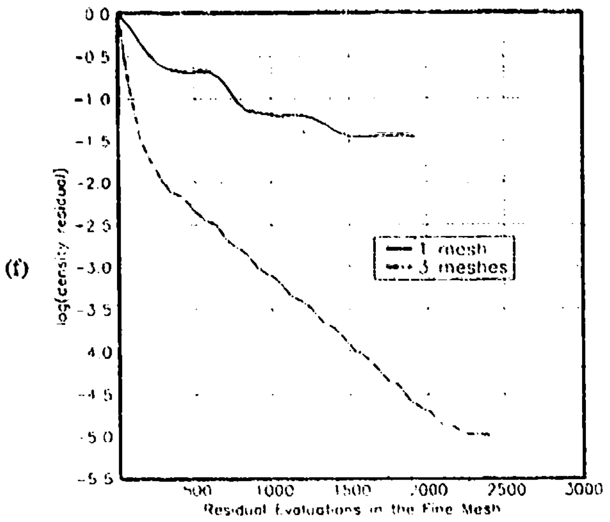
Flow About a Twin Engined Aircraft

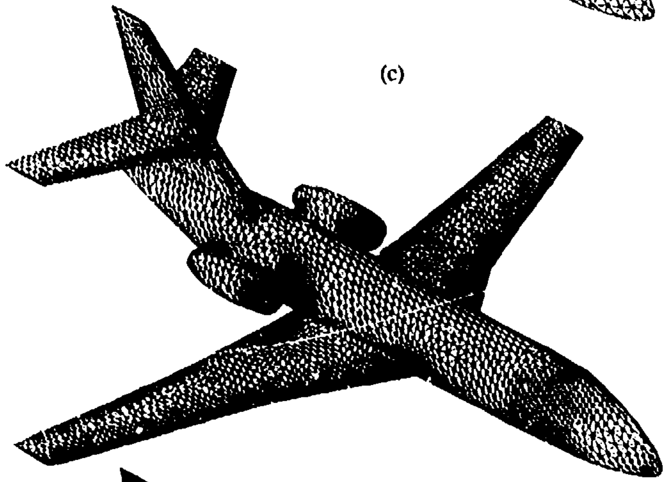
As an illustration of the performance of the multi-grid acceleration procedure for a realistic aeronautical configuration, a transonic flow about a twin engined Dassault Falcon has been computed. For the simulation, a free stream Mach number of 0.85 is assumed and the angle of attack is taken to be two degrees. Flow through conditions are imposed at the engines. Due to the assumed symmetry of the flow, the numerical computations were performed using only one half of the aircraft geometry. The representation of the boundary of the computational domain consists of 22 surface components and 47 curve components. The surface components used to represent the aircraft geometry are shown in figure 3.16a. Using this geometrical definition, three grids are generated, consisting of 863,967, 212,433 and 100,215 elements respectively. The surface triangulations for these grids are shown in figures 3.16b to 3.16d. A three stage time-stepping scheme and a double W cycle (*icycle* =2) is employed, with one pre- and one post-smoothing step used on each grid(*n1* =1, *n2* =1 and *n3* =1). A plot showing the of the value of the logarithm of the density residual versus the number of fine grid residual evaluations for both the fine grid only calculation and the multi-grid cycle is shown in figure 3.16f. The computed steady state distribution of pressure contours on the aircraft surface in shown in figure 3.16e. This solution was obtained in about 190 minutes of cpu time on a modern supercomputer using a single processor. For this example, the multi-grid version of the solver requires 89 storage locations per node. Thos compares very favourably with the 63 locations per node required by the flow solver on a single grid.

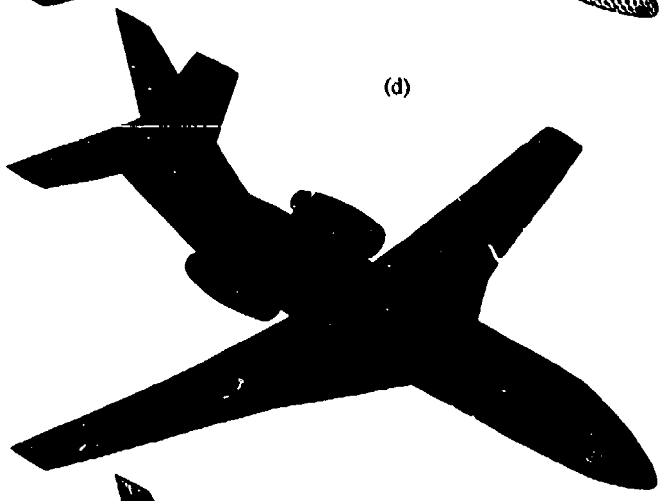
(a)











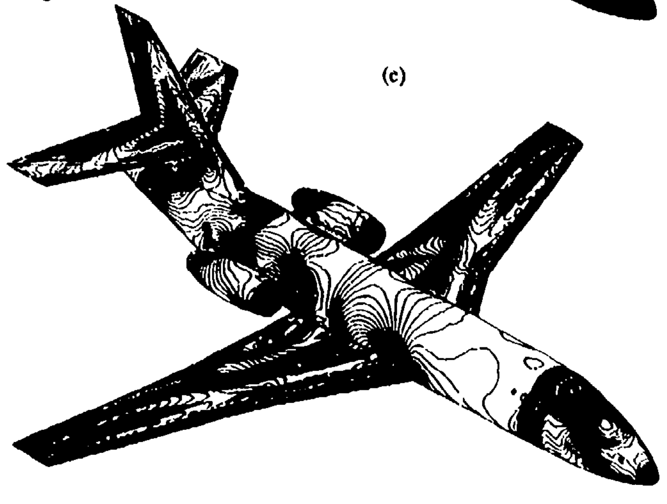


Figure 3.16
Flow past a Dassault Falcon at a free stream Mach number of 0.85 and at two degrees angle of attack (a) The geometry definition. (b) The first grid of 100,215 elements, 19,046 points (c) The second grid of 212,433 elements and 40,322 points (d) The third grid of 663,967 elements, 161,608 points. (e) Comparison of the convergence histories produced by using the fine grid alone and by using multi-grid.

4. DATA STRUCTURES

(This section has been written in collaboration with J. Bonet, Institute for Numerical Methods in Engineering, University College, Swansea SA2 8PP, UK)

From the previous section it is apparent that a successful implementation of the presented algorithm will require the use of data structures which enable certain sorting and searching operations to be performed efficiently. For instance, the generation front will require a data structure which allows for the efficient insertion/deletion of sides/faces and which also allows for the efficient identification of the sides/faces which intersect with a prescribed region in space.

The problem of determining the members of a set of n points which lie inside a prescribed subregion of an N dimensional space is known as *geometric searching*. Several algorithms have been proposed [37-40] which solve the above or equivalent problems with a computational expense proportional to log(n). The problem complexity increases considerably when, instead of considering points, one deals with finite size objects such line segments, triangles or tetrahedra. A common problem encountered here, namely *geometric intersection*, consists of finding the objects which overlap a certain subregion of the space being considered. Algorithms for solving this problem in two dimensions exist [41] and have been applied in determining the intersection between geometrical objects in the plane. To our knowledge, the only algorithm capable of solving this problem in three dimensions is based on the use of the alternate digital tree [42]. The particular application which motivated the development of this data structure was the implementation of the grid generation algorithm described in the previous section.

In what follows we shall describe an algorithm and associated data structure, called the alternating digital tree (ADT), which allows for the efficient solution of the geometric searching problem. It naturally offers the possibility of inserting and removing points and optimally searching for the points contained inside a given region. It is applicable to any number of dimensions, and is a natural extension of the so called digital tree search technique which is exhaustively in [43] for one dimensional problems. A procedure which allows treatment of any geometrical object in an N dimensional space as a point in a 2N dimensional space will be introduced; thereby allowing the proposed technique to be employed for the solution of geometric intersection problems.

4.1 Binary Tree Structures

Binary trees provide the basis for several searching algorithms, including the one to be presented here. It is therefore necessary to introduce some basic concepts and terminology related to binary tree structures. More detailed expositions can be found in [41,44].

Definition and Terminology

Tree structures provide a systematic way of storing a collection of data items which enables not only a quick access to the information stored, but also frequent insertions and deletions of items. This degree of flexibility requires the storage of data items in non-sequential locations of the computer memory. As figure 4.1a illustrates, to achieve this, each data item is extended by the addition of two integer values, known as the *left* and *right links*, and stored in what is known as a *node* of the tree. Each added link can either be equal to zero or equal to the position in memory where another node of the tree can be found. Hence, from one node of the tree it is possible to reach at most two other nodes. Moreover, in order to ensure that every node can be reached, these links must be such that for each node except one, known as the root, there is one and only one link pointing at it. This definition establishes a hierarchy of nodes: the root at the top level of the hierarchy points at 0, 1 or 2 nodes at the next level; each of these in turn points at other 0, 1 or 2 nodes at the next level of the hierarchy; and so forth. This hierarchical structure inspires the graphical representation shown in figure 4.1b for a simple tree comprising only eight nodes {A, B, C, D, E, F, G, H}.

Genealogical terms are normally used to describe the relative position of nodes in a tree: when a node points at a second node, the former is called the father of the latter, and this the son of the former node. A node without sons, that is, with both links blank, is called a terminal node, and the only node without a father is the root (node A in figure 4.1b). Given a node, the set of nodes formed by itself together with all its descendants constitutes a subtree of the main tree. For instance, in figure 4.1b the trees {C, D, E, F, G, H} and {E, G, H} are subtrees of the main tree rooted at C and E respectively.

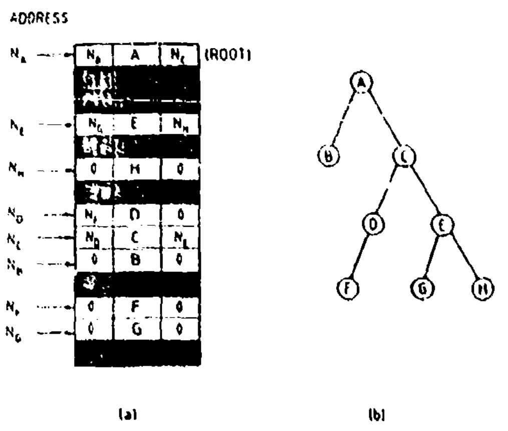


Figure 4.1
A simple binary tree and its storage in computer memory

Tree Traversal

To retrieve information stored in a given node requires knowledge of its location in memory, which is kept by its father. Hence, a node in the tree can only be examined or visited if all its ancestors are visited first. However, it is possible to systematically examine each node in such a way that every node is visited exactly once. Such an operation is known as traversing the tree and provides the basis for the searching methods discussed below. Although several algorithms can be found in the literature to traverse a binary tree [44], attention will be centred here on the so-called preorder traversal method. This technique is embodied in the following three steps:

1. *Visit the root of the current subtree*
2. *If the left link of the root is not zero then traverse the left subtree.*
3. *If the right link of the root is not zero then traverse the right subtree.*

The procedure determined by these three steps is clearly recursive, that is, steps 2 and 3 invoke again the algorithm which they define. In order to illustrate this process, consider again the tree shown in figure 4.1b; for this tree, the repeated application of the above algorithm yields the following sequence:

```
1. Traverse the tree {A, B, C, D, E, F, G, H}
   1.1. Visit A
   1.2. Traverse the tree {B}
        1.2.1. Visit B
        1.2.2. Skip
        1.2.3. Skip
   1.3. Traverse the tree {C, D, E, F, G, H}
        1.3.1. Visit C
        1.3.2. Traverse the tree {D, F}
               1.3.2.1. Visit D
               1.3.2.2. Traverse the tree {F}
                        1.3.2.2.1 Visit F
                        1.3.2.2.2 Skip
                        1.3.2.2.3 Skip
               1.3.2.3 Skip
        1.3.3. Traverse the tree {E, G, H}
               1.3.3.1. Visit E
               1.3.3.2. Traverse the tree {G}
                        1.3.3.2.1 Visit G
                        1.3.3.2.2 Skip
                        1.3.3.2.3 Skip
               1.3.3.3. Traverse the tree {H}
                        1.3.3.3.1 Visit H
                        1.3.3.3.2 Skip
                        1.3.3.3.3 Skip
```

Thus, the nodes of the tree in figure 4.1b in preorder are A, B, C, D, F, E, G and H.

We notice in the above algorithm that, before moving on to traverse the left subtree - step 2 in the previous algorithm - it is necessary to store the value of the right link, that is, the address of the right son, in order to enable the subsequent traversal of the right subtree. Moreover, whilst traversing the left subtree it is likely that additional right links will have to be stored. In fact, a list containing the addresses of all right subtrees encountered along the way which are yet to be traversed, must be kept and has to be continuously updated as follows. After visiting each node, the right link, if different from zero, is added to the list end if the left link is not zero the left subtree is traversed. When a zero left link is encountered, the last right link inserted in the list is retrieved, as well as removed, from the list and the subtree rooted at this address is traversed.

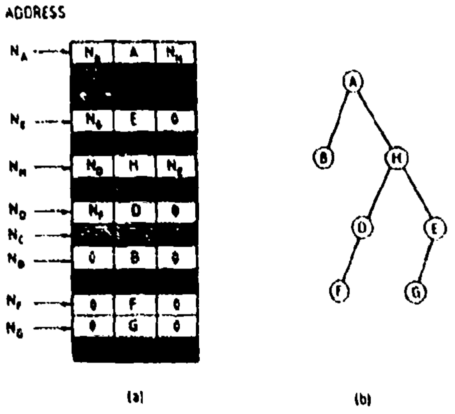


Figure 4.2
The deletion process.

This type of list, in which items are inserted one by one and extracted, also one at a time, in the reverse order, is known as a stack [44]. A stack consists of a linear array, or vector, together with an integer variable to record the number of items in the array. This variable, being initially zero, is increased by one every time an item is added to the stack and decreased by one when an item is extracted from it.

With the help of a stack, any recursive algorithm can be implemented without the need to use recursive routines. For instance, a non-recursive implementation of the traversal algorithm given above can be symbolically expressed as:

```
0.a Set root_address = address of the root node
0.b Set stack_size = 0
      1. Visit the node stored at root_address
      2. If right_link ≠ 0 then:
               Set stack_size = stack_size +1
               Set stack(stack_size) = right_link
         endif
      3a. If left_link ≠ 0 then:
               Set root_address = left_link
               go to 1
         endif
      3b. If left_link = 0 then:
               If stack_size ≠ 0 then:
                    Set root_address = stack(stack_size)
                    Set stack_size = stack_size - 1
                    go to 1
               endif
         endif
         If stack_size = 0      →      terminate the process
```

Inserting and Deleting Nodes

In order to add a new data item to a binary tree, a node containing the new item of information must be created and stored in a convenient memory location. The left and right links of this node are set to zero. If the current tree is empty, the new node becomes the root of the tree, otherwise the node must be inserted or linked to the existing tree. To achieve this, the tree is followed downwards, starting from the root and jumping from father to son, until a blank link is found. This link is then set to the memory position of the new node. When moving down the tree, a criterion must be provided at each node to chose between the left or right branches. This criterion determines the final position in the tree of the new node and, consequently, the shape of the tree itself.

Deleting a node from a binary tree is a straightforward operation if the undesired node is a terminal node; changing to zero the corresponding link of its father effectively 'prunes' the node from the tree and renders the memory occupied by it available for future uses. In the case of an intermediate node, the process becomes slightly more complicated since a gap can not be left in the tree. To overcome this problem, the unwanted node is replaced by a terminal node chosen from among its descendants. This operation can be carried out by modifying the links to suit the new structure of the tree and without moving the nodes from their memory positions. Figures 4.2a and 4.2b illustrate the deletion of node C from the tree shown in figures 4.1a and 4.1b and its replacement by node H.

If the application at hand demands frequent deletion and insertion, a memory book-keeping system is necessary for the efficient implementation of tree structures. This is required so that new nodes can be placed in the memory space released by the deletion of previous nodes. This problem can be solved by using a linked list structure to record all the available memory spaces. A linked list is a data structure that differs from the binary tree data structure described above in that every node has always only one link pointing at another node, and every node has always one link pointing at it. There are two exceptions, which are the *head* and the *end* nodes. The head is a node with no link pointing at it - the address of which is kept separately - and the end is a node with a blank link.

As shown in figure 4.3 the two data structures, binary tree and linked list, are updated simultaneously. Initially, the available memory is partitioned into cells of the correct size to store tree nodes. These cells, which contain no relevant information other than a single link, are then joined together to form a linked list. Every time a node needs to be inserted into the tree, the memory space required by this new tree node is generated by removing a node from the list (see figure 4.3b). Similarly, when a node is deleted from the tree it is added to the list (see figure 4.3c). Inserting and deleting nodes in the list always takes place at the head. To insert a node into the list, the link of the new node is set equal to the address of the head and the inserted node becomes the new head of the list. The deletion of the head node can be done by simply allowing its link to be the new head.

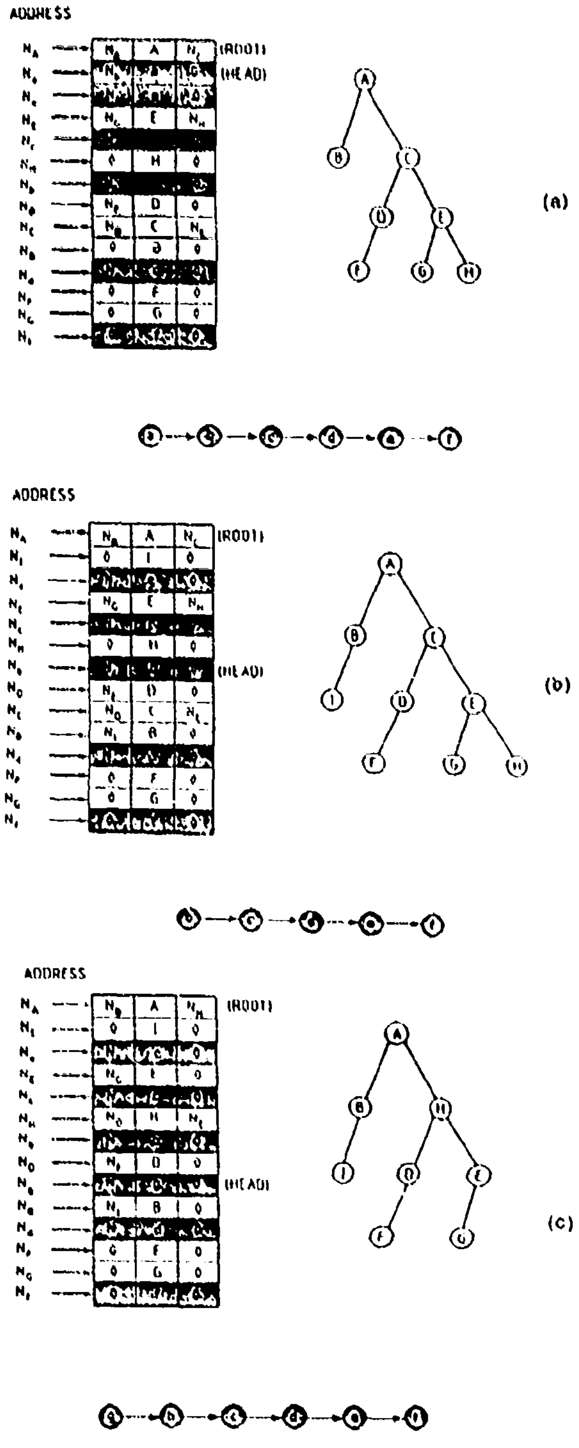


Figure 4.3
Binary tree/linked list configurations. (a) The tree of figure 4.1. (b) After the insertion of node I and using the storage released by node a. (c) After deleting node C.

4.2 The Alternating Digital Tree

Consider a set of n points in a N dimensional space (R^N) and assume for simplicity that the coordinate values of their position vectors $\{\underline{x}_1, \underline{x}_2 \dots \underline{x}_n\}$, after adequate scaling, vary within the interval [0,1). The aim of geometric searching algorithms is to select from this set those points that lie inside a given subregion of the space. To facilitate their representation, only rectangular - or 'hyper-rectangular' - regions will be considered, thereby allowing their definition in terms of the scaled coordinates of the lower and upper vertices as $(\underline{a}, \underline{b})$.

Comparing the coordinates of each point k with the vertex coordinates of a given subregion to check whether the condition $a_i \leq x^k_i \leq b_i$ is satisfied for i = 1, 2 ... N, would render the cost of the searching operation proportional to the number of points n. This computational expense, however, can be substantially reduced by storing the points in a binary tree, in such a way that the structure of the tree reflects the positions of the points in space. There exist several well known algorithms that will accomplish this effect for one dimensional problems; the most popular are the binary search tree and digital tree methods [41,43]. Binary search trees have been extended to N dimensional problems in [45], but the resulting tree structure, known as N-d *trees*, do not allow the efficient deletion of nodes. The algorithm presented here is a natural extension of the one dimensional digital tree algorithm and overcomes the difficulties encountered in N-d trees.

Definition and Node Insertion

Broadly speaking, an alternating digital tree can be defined as a binary tree in which a set of n points are stored following certain geometrical criteria. These criteria are based on the similarities arising between the hierarchical and parental structure of a binary tree and a recursive bisection process: each node in the tree has two sons, likewise a bisection process divides a given region into two smaller subregions. Consequently, it is possible to establish an association between tree nodes and subregions of the unit hypercube as follows: the root represents the unit hypercube itself; this region is now bisected across the x^1 axis and the region for which $0 \leq x^1 < 0.5$ is assigned to the left son and the region for which $0.5 \leq x^1 < 1$ is assigned to the right son; at each of these nodes the process is repeated across the x^2 direction as shown in figure 4.4. In a two dimensional space the process can be repeated indefinitely by chosing x^1 and x^2 directions in alternating order; similarly, in a general N dimensional space, the process can be continued by choosing directions $x^1, x^2, \dots x^N$ in cyclic order.

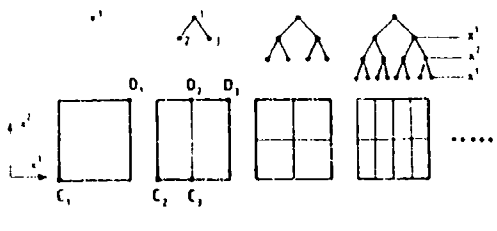


Figure 4.4
The relation between a binary tree and a bisection process.

Generally, if a node k at the hierarchy level m - the root being level 0 - represents a region $(\underline{c}_k, \underline{d}_k)$, the subregions associated to its left and right sons, $(\underline{c}_{kl}, \underline{d}_{kl})$ and $(\underline{c}_{kr}, \underline{d}_{kr})$ result from the bisection of $(\underline{c}_k, \underline{d}_k)$ by a plane normal to the j-th coordinate axis, where j is taken cyclically from the N space directions as:

$$j = 1 + \mathrm{mod}(m,N) \qquad (4.1)$$

and mod(m,N) denotes the remainder of the quotient of m over N. Hence $(\underline{c}_{kl}, \underline{d}_{kl})$ and $(\underline{c}_{kr}, \underline{d}_{kr})$ are obtained as:

$$c_{kl}{}^i = c_k{}^i,\ d_{kl}{}^i = d_k{}^i \quad \text{for } i \neq j \text{ and } \quad c_{kl}{}^j = c_k{}^j,\ d_{kl}{}^j = \frac{1}{2}(c_k{}^j + d_k{}^j) \qquad (4.2a)$$

$$c_{kr}{}^i = c_k{}^i,\ d_{kr}{}^i = d_k{}^i \quad \text{for } i \neq j \text{ and } \quad c_{kr}{}^j = \frac{1}{2}(c_k{}^j + d_k{}^j),\ d_{kr}{}^j = d_k{}^j \qquad (4.2b)$$

This correlation between nodes and subdivisions of the unit hypercube allows an ADT to be further defined by imposing that each point in the tree should lie inside the region corresponding to the node where it is stored. Consequently, if node k of an ADT structure contains a point with coordinates $\underline{x}_k$, the following condition must be satisfied:

$$c_k{}^i \leq x_k{}^i < d_k{}^i \qquad \text{for } i = 1,2 \dots N \qquad (4.3)$$

Due to this additional requirement there exists only one possible way in which a new point can be inserted in the tree. As discussed in the previous section the tree is followed downwards until an unfilled position where the node can be placed is found. During this process, however, left or right branches are now chosen according to whether the new point lies inside the region related to the left or right sons, thereby ensuring that condition (4.3) is satisfied.

Given a predetermined set of n points, an ADT structure can be built by placing anyone point at the root and then inserting the remaining points in consecutive order according to the algorithm described above. This is illustrated in figure 4.5 for a set of 5 points {A,B,C,D,E}. The shape of the tree obtained in this way depends mainly on the spatial distribution of the points and somewhat on the order in which the points were inserted. The cost of operations like node insertion/deletion and geometric searching depends strongly on the shape of the tree; generally poor performances are to be expected from highly degenerated trees (see figure 4.6), whereas well balanced trees (see figure 4.7), as those obtained for fairly uniform distributions of points, will result in substantial reductions of the searching cost. In these cases the average number of levels in the tree, and therefore the average cost of inserting a new point, becomes proportional to log(n); clearly a considerable cost if compared with the cost of storing the poir a sequential list, but fully justifiable in view of the reduction in ching costs that ADT structures will provide.

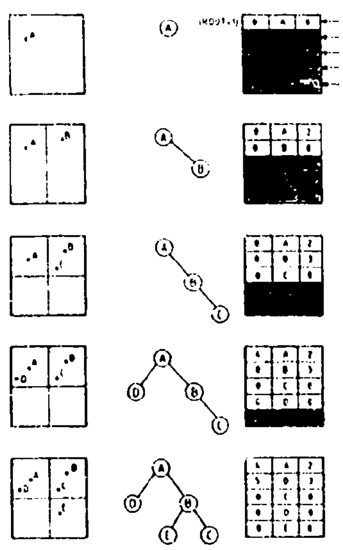


Figure 4.5
Building an ADT by successive insertion.

Geometric Searching

Consider now a set of points stored in an ADT structure. The fact that condition (4.3) is satisfied by every point provides the key to the efficient solution of a geometric searching problem. To illustrate this, note first that the recursive structure of the bisection process described above implies that the region related to a given node k consequently, all points stored in these nodes must also lie inside the region represented by node k. For instance, all points in the ADT structure are stored in nodes descended from the root and, clearly, all of them lie inside the unit hypercube - the region associated with the root. Analogously, the complete set of points stored in any subtree is inside the region represented by the root of the subtree

This feature can be effectively used to reduce the cost of a geometric searching process by checking, at any node k, the intersection between the searching range ($\underline{a}$, $\underline{b}$) and the region represented by node k, namely ($\underline{c}_k$, $\underline{d}_k$). If these two regions fail to overlap, then the complete set of points stored in the subtree rooted at k can be disregarded from the search, thus avoiding the need to examine the coordinates of every single point.

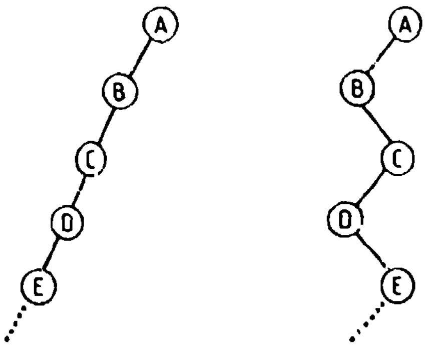


Figure 4.6
Degenerated trees.

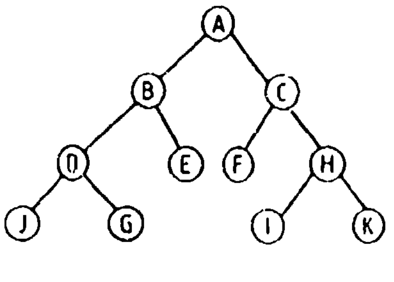


Figure 4.7
A well balanced tree.

Consequently, a systematic procedure to select the points that lie inside a given searching range ($\underline{a}$, $\underline{b}$) can be derived from the traversal algorithm previously presented. Now the generic operation 'visit the root' can be re-interpreted as checking whether the point stored in the root falls inside the searching range. Additionally, the left and right subtrees need to be traversed only if the regions associated with their respective root nodes intersect with the range. Accordingly, a geometric searching algorithm emerges in a recursive form as:

1. *Check whether the coordinates of the node stored in the root, say* $\underline{x}_k$, *are inside* ($\underline{a}$, $\underline{b}$) *i.e. check whether* $a^i \le x_k^i < b^i$ *for* $i = 1,2 \ldots N$.
2. *If the left link of the root is not zero and the region* ($\underline{c}_{kl}$, $\underline{d}_{kl}$) *overlaps with* ($\underline{a}$, $\underline{b}$) *i.e. if* $d_{kl}^i \geq a^i$ *and* $c_{kl}^i \le b^i$ *for* $i = 1,2 \ldots N$, *search the left subtree.*
3. *If the right link of the root is not zero and the region* ($\underline{c}_{kr}$, $\underline{d}_{kr}$) *overlaps with* ($\underline{a}$, $\underline{b}$) *i.e. if* $d_{kr}^i \geq a^i$ *and* $c_{kr}^i \le b^i$ *for* $i = 1,2 \ldots N$, *search the right subtree.*

In order to illustrate this process, consider the set of points and the searching range shown in figure 4.8a and the corresponding alternating digital tree depicted in figure 4.8b. For this simple example, the algorithm given above results in the following sequence of steps:

```
Search the tree {A,B,C,D,E,F,G,H}:
    1. Check if $a^i \le x_A^i \le b^i$ for $i = 1,2$
    2. Since $d_B^i \ge a^i$ and $c_B^i \le b^i$ search the tree
       {B,C,D,E}:
        2.1 Check if $a^i \le x_B^i \le b^i$
        2.2. Since $d_C^i \ge a^i$ and $c_C^i \le b^i$ search the tree
             {C,E}:
            2.2.1. Check if $a^i \le x_C^i \le b^i$
            2.2.2. Skip (left link is zero)
            2.2.3. Skip ($c_E^1 > b^1$)
        2.3. Skip ($c_D^2 > b^2$)
    3. Skip ($c_F^1 > b^1$)
```

Again a 'non-recursive' implementation of this algorithm can be achieved using a stack in a very similar way to that previously described for the traversal algorithm.

Note that, with this technique, only the coordinates of points A,B and C are actually examined, the rest being immediately disregarded in view of their position in the tree. In general, only those points stored in nodes with associated regions overlapping ($\underline{a}$, $\underline{b}$) will be checked during the searching process.

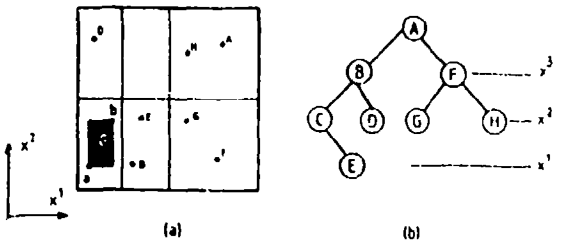


Figure 4.8
The searching problem in R^2.

4.3 Geometric Intersection

Geometrical intersection problems can be found in many applications; for instance, a common problem that may emerge in contact algorithms [46], hidden line removal applications or in the advancing front grid generation algorithm presented in section 3, is to determine from a set of three noded triangular elements those which intersect with a given line segment. Similar problems, involving other geometrical objects, are encountered in a wide range of geometrical applications. In general, a geometric intersection problem consists of finding from a set of geometrical objects those which intersect with a given object. If every one-to-one intersection is investigated, the solution of these problems can become very expensive, especially when complex objects such as curves or surfaces are involved. Fortunately, many of these one-to-one intersections can be quickly discarded by means of a simple comparison between the coordinate limits of every given pair of objects. For instance, a triangle with x-coordinate varying from 0.5 and 0.7 cannot intersect with a segment with x-coordinate ranging from 0.1 to 0.3. Generally, the intersection between two objects in the N dimensional Euclidean space, requires each of the N pairs of coordinate ranges to overlap. Consider for instance the intersection problem between triangular facets and a target straight line segment in R^3; then, if ($\underline{x}_{k,min}$, $\underline{x}_{k,max}$) are the coordinate limits of element k and ($\underline{x}_{0,min}$, $\underline{x}_{0,max}$) are the lower and upper limits of the target segment (see figure 4.9), an important step towards the solution of a geometric intersection problem is to select those which satisfy the inequality:

$$\begin{aligned} x_{k,min}^i &\le x_{0,max}^i \\ & \qquad\qquad \text{for } i = 1,2 \ldots N \qquad (4.4) \\ x_{k,max}^i &\ge x_{0,min}^i \end{aligned}$$

The cost of checking condition (4.4) for every element grows proportionally to n, and for very numerous sets may become prohibitive. This cost, however, can be substantially reduced by using a simple device whereby the process of selecting those elements which satisfy condition (4.4) can be interpreted as a geometric searching problem. Additionally, since the number of elements that satisfy condition (4.4) will normally be much smaller than n, the cost of determining which of these intersects with the target segment becomes affordable.

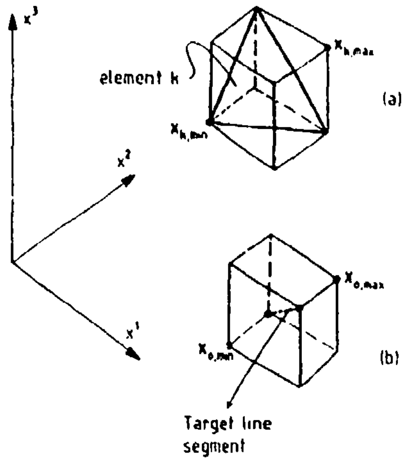


Figure 4.9
The definition of coordinate limits for triangular elements and straight line segments.

In order to interpret condition (4.4) as a geometric searching problem, it is first convenient to assume that all the elements to be considered lie inside a unit hypercube - a requirement that can be easily satisfied through adequate scaling of the coordinate values. Consequently, condition (4.4) can be re-written as:

$$\begin{aligned} 0 \le x_{k,min}{}^1 &\le x_{0,max}{}^1 \\ &\vdots \\ 0 \le x_{k,min}{}^N &\le x_{0,max}{}^N \\ x_{0,min}{}^1 \le x_{k,max}{}^1 &\le 1 \\ &\vdots \\ x_{0,min}{}^N \le x_{k,max}{}^N &\le 1 \end{aligned} \qquad (4.5)$$

Consider now a given object k in R^N with coordinate limits $\underline{x}_{k,min}$, and $\underline{x}_{k,max}$; combining this two sets of coordinate values, it is possible to view an object k in R^N as a point in R^{2N} with coordinates $\underline{x}_k{}^i$ for i = 1,2 ... 2N defined as (see figure 4.10):

$$\underline{x}_k = [\, x_{k,min}{}^1, \dots x_{k,min}{}^N, x_{k,max}{}^1, \dots x_{k,max}{}^N]^T \qquad (4.6)$$

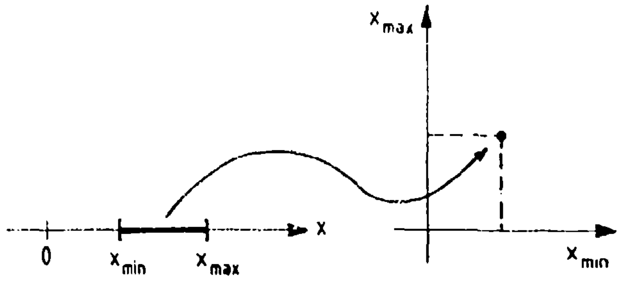


Figure 4.10
The representation of a region in R^1 as a point in R^2.

Using this representation of a given object k, condition (4.5) becomes simply:

$$a^i \le x_k{}^i \le b^i \qquad \text{for } i = 1,2 \dots 2N \qquad (4.7)$$

where $\underline{a}$ and $\underline{b}$ can be interpreted as the lower and upper vertices of a 'hyper-rectangular' region in R^{2N} and, recalling (4.5), their components can be obtained in terms of the coordinate limits of the target object (see figure 4.11) as:

$$\underline{a} = [\, 0, \dots 0, x_{0,max}{}^1, \dots x_{0,max}{}^N]^T \qquad (4.8a)$$
$$\underline{b} = [\, x_{0,min}{}^1, \dots x_{0,min}{}^N, 1, \dots 1]^T \qquad (4.8b)$$

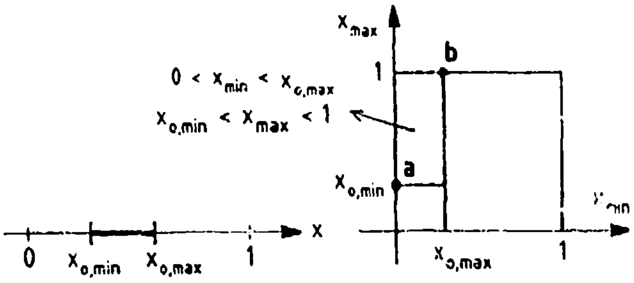


Figure 4.11
The intersection problem in R^1 as a searching problem in R^2.

Consequently, the problem of finding which objects in R^N satisfy condition (4.4) becomes equivalent to a geometric searching problem in R^{2N} i.e. obtaining the points $\underline{x}_k$ which lie inside the region limited by $\underline{a}$ and $\underline{b}$. Once this subgroup of elements has been selected, the intersection of each one of them with the target object must be checked to complete the solution of the geometric intersection problem.

4.4. The Use of the ADT for Grid Generation

It is obvious from the advancing front algorithm described in section 3 that operations such as searching for the points inside a certain region of the space and determining intersections between geometrical objects - in this case sides and faces - will be performed very frequently. The complexity of the problem is increased by the fact that the set of faces forming the generation front changes continuously as new faces need to be inserted and deleted during the process. Clearly, for grids consisting of a large number of elements the cost of performing this operations can be very important.

A successful implementation of the above algorithms has been accomplished by making extensive use of the ADT data structure. For instance, the algorithm of section 3.5 for tetrahedra generation employs two tree structures; one for the faces in the front and the other for the sides defined by the intersection between each pair of faces in the front (see figure 3.11). This combination allows a high degree of flexibility and the operations of insertion, deletion, geometric searching and geometric intersection can be performed optimally. The overall computational performance of the algorithm is demonstrated by generating tetrahedral grids, using the above method, for a unit cube (see figure 4.12). Different numbers of elements have been obtained by varying the grid size. In figure 4.12 the computer time required on a VAX 8700 machine has been plotted against the number NE of elements generated. It can be observed that a typical NE*log(NE) behaviour is attained. Using this approach grids containing up to one million elements have been generated and no degradation in the performance has been detected.

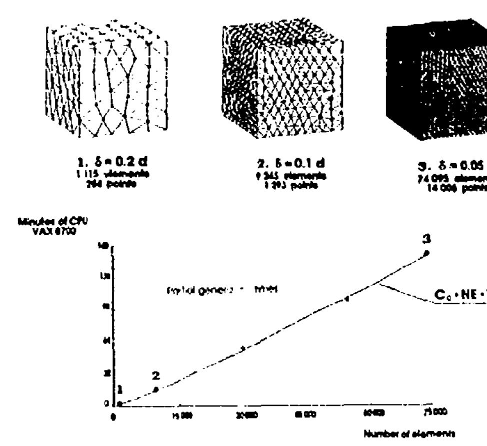


Figure 4.12
Grid generation cpu times.

5. ADAPTIVITY FOR STEADY STATE PROBLEMS

The procedures described above allow for the computation of an initial approximation to the steady state solution of a given problem. This approximation can generally be improved by adapting the grid in some manner. Here, we follow the approach of using the computed solution to predict the desired characteristics (i. e. element size and shape) for a new, adapted grid. The ultimate aim of the adaptation procedure is to predict the characteristics of the optimal grid. This can be defined as the grid in which the number of degrees of freedom required to achieve a specified level of accuracy is a minimum. Alternatively, it can be interpreted as the grid in which a given number of degrees of freedom are distributed in such a manner that the highest possible solution accuracy is achieved. In practical situations however, there are several factors which make the achievement of such optimal grids extremely difficult. Some of these factors are:

i) The concept of optimality is intimately linked to that of accuracy, which is not uniquely defined. Hence optimality of a grid needs to be defined with respect to a given norm or measure of the error. An additional inconvenience related to the measure of accuracy, in the present context, arises from the fact that we are attempting to solve a coupled set of non linear partial differential equations and, therefore, a rigourous measure of the error should involve all the relevant variables.

ii) For linear elliptic operators, as we have shown in Section 1, Galerkin finite element algorithms are readily derived which guarantee that the approximation obtained is the most accurate amongst all the possible approximations within the trial space of functions. Here, accuracy is defined with respect to a norm implied by the operator itself (the energy norm). For the Euler equations, however, such an energy norm does not exist and no numerical schemes are known which possess this optimality property.

iii) This best approximation property means that the error of the computed solution, measured in the energy norm, is bounded above by that of the exact interpolant. i.e. the approximation in the space of current trial functions which has exact nodal values. Using results of interpolation theory [47], it is then possible to produce rigourous bounds on the error of the numerical approximation. These results are based on certain regularity assumptions on the solution, which for the Euler equations will be invalid in the vicinity of discontinuities in the flow.

iv) Finally, the error estimates produced are based on the computed solution. As this is only an approximate solution, such error estimates will only be as good as the computed solution. This means that, even in the best situation, the optimal grid will only be achieved in the asymptotic limit. i.e. when the solution is so good that the computed error becomes very reliable.

In view of these observations and limitations, we have made an attempt to develop a heuristic adaptive strategy. This strategy uses error estimates which are based upon concepts from interpolation theory. The possible presence of discontinuities in the solution is taken into account and, in addition, the procedure provides information about any directionality which may be present in the solution. The advantages of using directional error indicators become apparent when we consider the nature of the solutions to be computed involving flows with shocks, contact discontinuities etc. Such features can be most economically represented on grids which are stretched in appropriate directions. Although, these error estimates have no associated mathematical rigour considerable success has been achieved with their use in practical situations.

The computed error, estimated from the current solution, is transformed into a spatial distribution of 'optimal' grid spacings which are interpolated using the current grid. The current grid is then modified with the objective of meeting these 'optimal' distribution of grid characteristics as closely as possible. Three alternative procedures will be discussed here for performing the grid adaption. The resulting grid is employed to produce a new solution and this procedure can repeated several times until the user is satisfied with the quality of the computed solution.

5.1 Error Indicator in 1D

The development of a method for error indication is considerably simplified if we restrict consideration to problems involving a single scalar variable. For this reason, when solving the Euler equations, a key variable is identified and then the grid adaptation is based on an error analysis for that variable alone. The choice of the best variable to use as a key variable remains an open question, but the the Mach number has been adopted for the computations reported in these notes.

Consider first the one dimensional situation in which the exact values of the key variable σ are approximated by a piecewise linear function $\hat{\sigma}$. The error E is then defined as

$$E = \sigma(x^1) - \hat{\sigma}(x^1) \qquad (5.1)$$

We note here that if the exact solution is a linear function of x^1 then the error will vanish. This is because our approximation has been obtained using piecewise linear finite element shape functions. Moreover, if the exact solution is not linear, but is smooth, then it can be represented, to any order of precision, using polynomial shape functions.

To a first order of approximation, the error E can be evaluated as the difference between a quadratic finite element solution $\hat{\sigma}$ and the linear computed solution. To obtain a piecewise quadratic approximation one could obviously solve a new problem using quadratic shape functions. This procedure however, although possible, is not advisable as it would be even more costly than the original computation. An alternative approach for estimating a quadratic approximation from the linear finite element solution is therefore employed. Assuming that the nodal values of the quadratic and linear approximations coincide i.e. the nodal values of E are zero, a quadratic solution can be constructed on each element, once the value of the second derivative is known. Thus the variation of the error E within an element e can be expressed as

$$E_e = \frac{1}{2}\zeta(h_e - \zeta)\left.\frac{d^2\hat{\sigma}}{dx^{1\,2}}\right|_e \qquad (5.2)$$

where ζ denotes a local element coordinate and h_e denotes the element length. A procedure for estimating the second derivative of a piecewise linear function is described below.

The root mean square value E_e^{RMS} of this error over the element can be computed as

$$E_e^{RMS} = \left\{\int_0^{h_e}\frac{E_e^2}{h_e}\,d\zeta\right\}^{1/2} = \frac{1}{\sqrt{120}}h_e^2\left|\frac{d^2\hat{\sigma}}{dx^{1\,2}}\right|_e \qquad (5.3)$$

where | . | stands for absolute value.

We define the 'optimal' grid, for a given degree of accuracy, as the grid in which this root mean square error is equal over each element. In the present context, this requirement may be regarded as being somewhat arbitrary. However, it has been shown [48] that the requirement of equidistribution of the error leads to optimal results when applied to certain elliptic problems. This requirement is therefore written as

$$h_e^2 \left| \frac{d^2\hat{\sigma}}{dx^{12}} \right| = C \qquad (5.4)$$

where C denotes a positive constant.

Finally, the requirement of equation (5.4) suggests that the 'optimal' spacing δ on the new adapted grid should be computed according to

$$\delta^2 \left| \frac{d^2\hat{\sigma}}{dx^{12}} \right| = C \qquad (5.5)$$

5.2 Recovery of the Second Derivatives

The first derivative of the computed solution on a grid of linear elements will be piecewise constant and discontinuous across elements. Therefore, straightforward differentiation of $\hat{\sigma}$ leads to a second derivative which is zero inside each element and is not defined at the nodes. However, by using a recovery process, based upon a variational or weighted residual statement [12], it is possible to compute nodal values of the second derivatives from element values of the first derivatives of $\hat{\sigma}$.

To illustrate this process, consider a one dimensional domain $0 < x^1 < L$ which has been discretised into (n-1) linear two noded finite elements. The piecewise linear distribution of the computed solution $\hat{\sigma}$ is expressed as

$$\hat{\sigma} = \sum_{J=1}^{n} N_J \hat{\sigma}_J \qquad (5.6)$$

where N_J is the standard linear finite element shape function [12] associated to node J. Similarly, a piecewise linear approximation to the distribution of the second derivative, which we seek to determine, can be written as

$$\frac{d^2\hat{\sigma}}{dx^{12}} = \sum_{J=1}^{n} N_J \left. \frac{d^2\hat{\sigma}}{dx^{12}} \right|_J \qquad (5.7)$$

The nodal values of the second derivative may be computed from the approximate variational requirement that

$$\sum_{J=1}^{n} \left(\int_\Omega N_J N_K \, d\Omega \right) \left. \frac{d^2\hat{\sigma}}{dx^{12}} \right|_J = -\int_\Omega \left(\sum_{J=1}^{n} \frac{dN_J}{dx^1} \hat{\sigma}_J \right) \frac{dN_K}{dx^1} d\Omega$$

$$- \left(\frac{d\hat{\sigma}}{dx^1} N_K \right)_{x^1=0} + \left(\frac{d\hat{\sigma}}{dx^1} N_K \right)_{x^1=L} \quad K = 1, \ldots, n \qquad (5.8)$$

The values of the derivatives at the two end points can be inserted, if known, or can be taken to be equal to the constant value of the derivative in the adjacent elements. The resulting set of algebraic equations can be solved, in a few iterations, by using a Jacobi procedure [16] or alternatively, the consistent mass matrix appearing on the left hand side of equation (5.7) can be lumped, thus yielding a diagonal system of equations. Numerical results obtained to date do not indicate any significant differences in the grids produced by using these two approaches.

5.3 Extension to Multi-Dimensions

Following the process described above, nodal values of the second derivative can be obtained from the approximate solution on the current grid. The use of expression (5.5) then yields directly a nodal value of the 'optimal' spacing for the new grid.

Expression (5.5) can be directly extended to the N dimensional case by writing the quadratic form

$$\delta_\beta^2 \left(\sum_{i,j=1}^{N} m^{ij} \beta^i \beta^j \right) = C \qquad (5.9)$$

where β is an arbitrary unit vector, δ_β is the spacing along the direction of β, and m^{ij} are the components of a NxN symmetric matrix of second derivatives

$$m^{ij} = \frac{\partial^2\hat{\sigma}}{\partial x^i \partial x^j} \qquad (5.10)$$

These derivatives are computed, at each node of the current grid, by using the N dimensional equivalent of the procedure presented in the previous section. The meaning of equation (5.9) is graphically illustrated in figure 5.1 which shows how the value of the spacing in the β direction can be obtained as the distance from the origin to the point of intersection of the vector β with the surface of an ellipsoid. The directions and lengths of the axes of the ellipsoid are the principal directions and eigenvalues of the matrix **m** respectively.

Several alternative procedures exist for modifying an existing grid in such a way that the requirement expressed by equation (5.8) is more closely satisfied. Three such methods will be described here. In the first procedure, called *grid enrichment*, the nodes of the current grid are kept fixed but some new nodes/elements are created. In the second procedure, referred to as *grid movement*, the total number of elements and nodes remains fixed but their position is altered. Finally, in the *adaptive regriding* algorithm, the grid adaption is accomplished by completely regenerating a new grid using the grid generation algorithm presented in section 3.

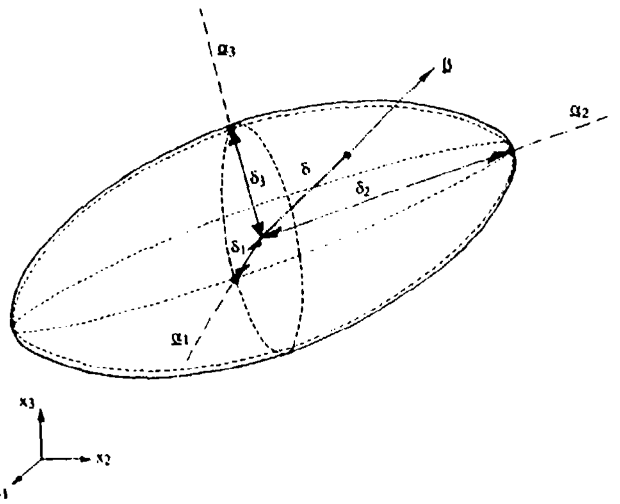


Figure 5.1
The determination of the value of the spacing δ along the direction β.

5.4 Grid Enrichment

In order to adapt a grid using grid enrichment, a sweep over all the sides in the grid is made and the 'optimal' spacing in the direction of each side is computed according to expression (5.9). For each side, the matrix **m** is taken to be the average of its value at the two nodes of the side. The enrichment procedure consists of introducing an additional node for each side for which the calculated spacing is less than the length of the side. For interior sides, this additional node is placed at the mid-point of the side, whereas for boundary sides, it is necessary to refer to the boundary definition and to ensure that the new node is placed on the true boundary. When any side is subdivided in this manner, the elements associated with that side will also need to be subdivided in order to preserve the consistency of the final grid. Figure 5.2 illustrates the three possible ways in which this element subdivision might have to be performed in two dimensions. The number of sides to be refined depends on the choice of the constant C in equation 5.9. To avoid excessive refinement in the vicinity of discontinuities, a minimum threshold value for the computed spacing can be used. When the grid enrichment procedure has been completed, the values of the unknowns at the new nodes are linearly interpolated from the original grid and the solution algorithm is re-started. This procedure has been successfully implemented in two and three dimensions and several impressive demonstrations of the power of this technique have been made. [8,19,49,50].

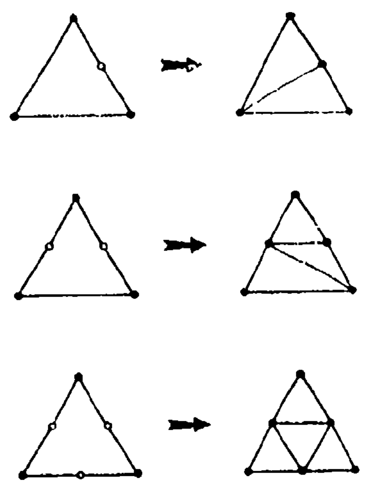


Figure 5.2
The grid enrichment process showing three possible refinement cases for a triangular element.

The application of the enrichment procedure in the solution of a two dimensional example is illustrated in figure 5.3. The problem solved is a Mach 8.15 flow past a double ellipse configuration at 30⁰ angle of attack. The initial grid and two adaptively enriched grids are shown together with the computed Mach number solutions. The application of the enrichment algorithm in three dimensions is shown in figure 5.4. The inviscid flow past a 30⁰ wedge is solved. The free stream Mach number is 3. This is a two dimensional problem computed on a three dimensional grid. Two views of the initial grid and solution are shown. A single application of the enrichment algorithm produces the grid and solution which are also displayed in figure 5.4.

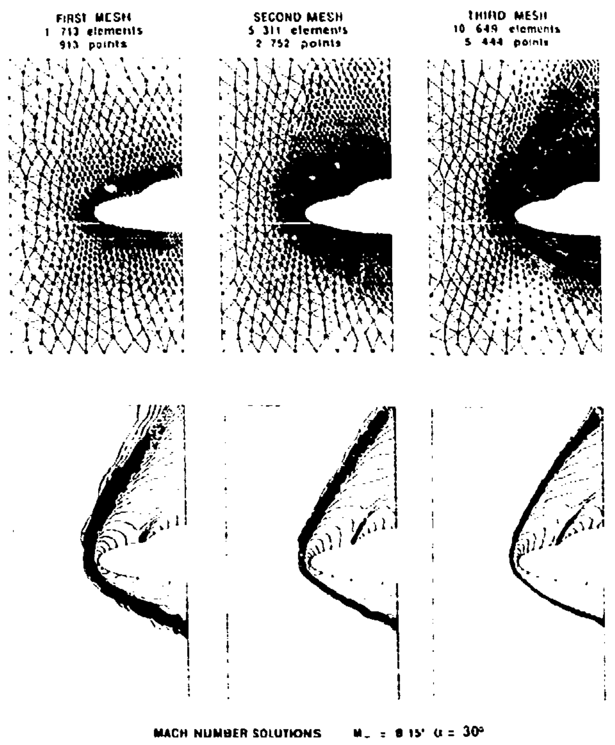


Figure 5.3
Supersonic flow past a double ellipse at a free stream Mach number of 8.15 and at an angle of attack of thirty degrees showing a sequence of grids and solutions obtained following the use of adaptive enrichment.

It can be observed, from the examples presented, how the quality of the solution is significantly improved by the application of the enrichment procedure. The main drawback of the approach is that the number of elements increases considerably following each application of the procedure. This means that, in the simulation of practical three dimensional problems, only a small number of such adaptations can be contemplated.

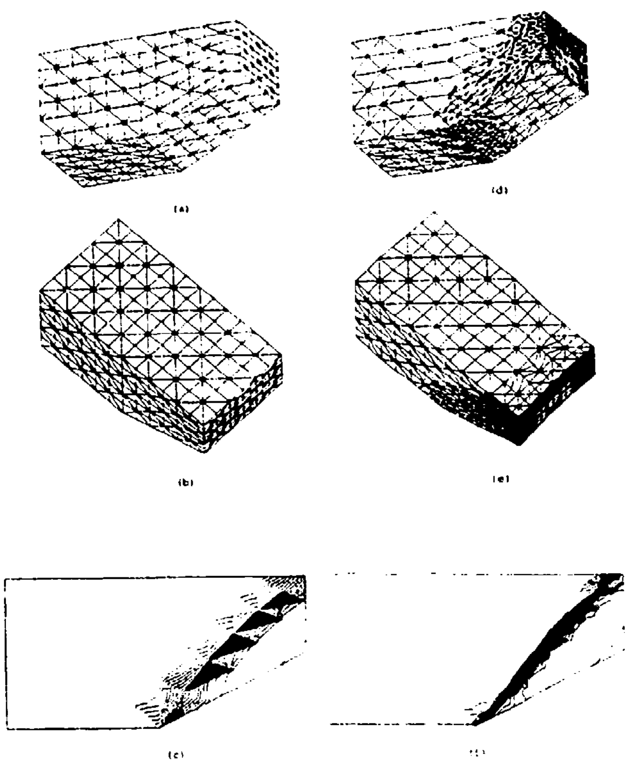


Figure 5.4
Supersonic flow past a wedge of angle thirty degrees in three dimensions.
(a) (b)Views of the initial grid. (c) The computed density contours.
(d) (e) Views of the enriched grid. (f) The computed density contours on the enriched grid.

5.5 Grid Movement

For the grid movement algorithm, the element sides are considered as springs of prescribed stiffness and the nodes are moved until the spring system is in equilibrium. Consider two adjacent nodes J and K as shown in figure 5.5. The force $\underline{f}_{JK}$ exerted by the spring connecting these two nodes can be taken to be

$$\underline{f}_{JK} = C_{JK} \left(\underline{r}_J - \underline{r}_K \right) \tag{5.11}$$

where C_{JK} is the stiffness of the spring and $\underline{r}_J$ and $\underline{r}_K$ are the position vectors of nodes J and K respectively. Assuming that

$$h = \left| \underline{r}_J - \underline{r}_K \right| \tag{5.12}$$

the adaptation requirement of equation (5.9) will be satisfied if the spring stiffnesses are defined as

$$C_{JK} = h \left| \sum_{i,j=1}^{N} m^{ij} n_{JK}{}^{i} n_{JK}{}^{j} \right| \tag{5.13}$$

Here $\underline{n}_{JK}$ is the unit vector in the direction of the side joining nodes J and K. For equilibrium, the sum of spring forces at each node should be equal to zero. The assembled system can be brought into equilibrium by simple iteration. In each iteration, a loop is performed over all the interior nodes and new nodal coordinates are calculated according to the expression

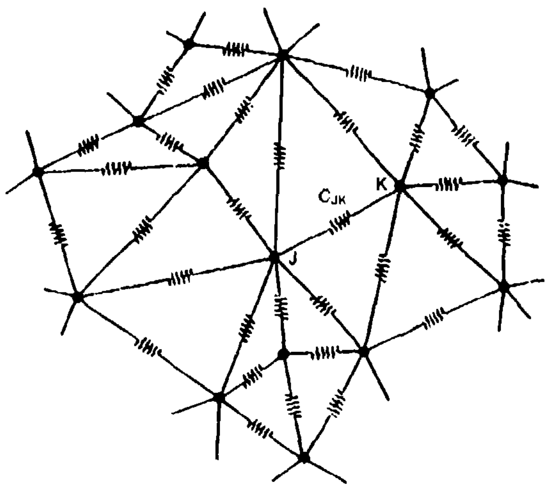


Figure 5.5
The grid movement algorithm in which element sides are replaced by springs.

$$\underline{r}_J^{NEW} = \frac{\sum_{K=1}^{S_J} C_{JK}\, \underline{r}_K}{\sum_{K=1}^{S_J} \underline{r}_K} \qquad (5.14)$$

where the summation extends over the number of nodes, S_J, which surround node J. Sufficient convergence is normally achieved after three to five passes through this procedure.

This technique will not necessarily produce grids of better quality, as badly formed elements can appear in regions (such as shocks) in which the spring coefficients C_{JK} vary rapidly over a short distance. To avoid this problem, the definition of the value of C_{JK} given in equation is (5.13) can be replaced by an expression of the form

$$C_{JK}^{MOD} = 1 + \frac{A\, C_{JK}}{B + C_{JK}} \qquad (5.15)$$

This can be regarded as a blending function definition for the spring stiffnesses and it has been constructed so as to ensure that, with a suitable choice for the constants A and B, excessively small or excessively large element sizes are avoided. This, in turn means that grids of acceptable quality will be produced. More sophisticated procedures for controlling the quality of the grid during movement can also be devised [51] and grid movement algorithms have been successfully used in two and three dimensional flow simulations on both structured and unstructured grids [15,51,52].

The grid movement algorithm described has been applied to the problem of flow past a double ellipse configuration which has been treated previously. Figure 5.6 shows the solutions produced following two grid adaptations. It can be seen that the improvement obtained after the second adaptation is minor. This is because the algorithm does not allow for the creation of new nodes and so the quality of the final solution is very much dependent on the topology of the initial grid. This is a major drawback of the grid movement strategy. A possible remedy to this problem is to combine grid enrichment and grid movement procedures. This is demonstrated in figure 5.7 which shows the application of the movement procedure to the final enriched grid of figure 5.3.

5.6 Adaptive Regriding

The basic idea of the adaptive regriding technique is to use the computed solution to provide information on the spatial distribution of the grid parameters. This information will be used by the grid generator described in section 3 to generate a completely new adapted grid for the problem under investigation.

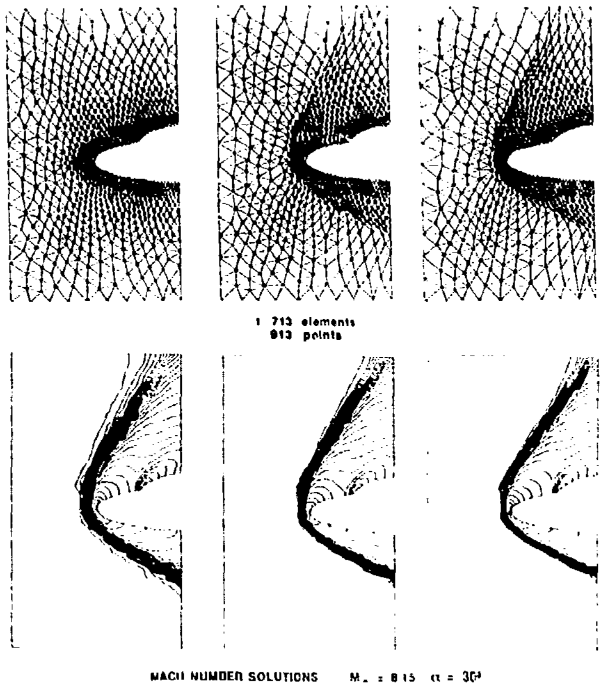


Figure 5.6
Supersonic flow past a double ellipse at a free stream Mach number of 8.15 and at an angle of attack of thirty degrees showing a sequence of grids and solutions obtained following the use of adaptive grid movement

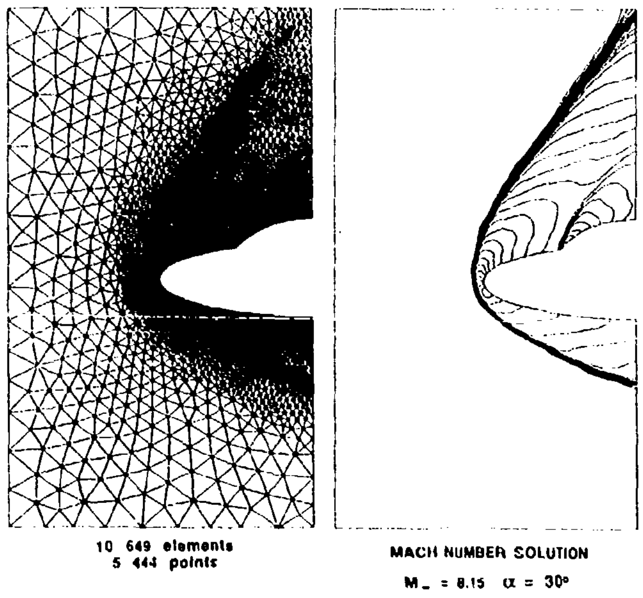


Figure 5.7
Supersonic flow past a double ellipse at a free stream Mach number of 8.15 and at an angle of attack of thirty degrees showing the solutions obtained following the application of adaptive grid movement to a previously enriched grid.

The 'optimal' values for the grid parameters are calculated at each node of the current grid. The directions $\underline{\alpha}_i$; i=1, ..., N, are taken to be the principal directions of the matrix m. The corresponding grid spacings are computed from the eigenvalues e_i; i=1, ..., N, as

$$\delta_i = \left\{\frac{C}{e_i}\right\}^{1/2} \qquad \text{for} \quad i=1, ..., N \qquad (5.16)$$

The spatial distribution of the grid parameters is defined when a value is specified for the constant C. The total number of elements in the adapted grid will depend upon the choice of this constant. For smooth regions of the flow, this constant will determine the value of the root mean square error in the key variable that we are willing to accept. Therefore this constant should be decreased each time a new grid adaption is performed. On the other hand, solutions of the Euler equations are known to exhibit discontinuities. At such discontinuities, the root mean square error will always remain large and therefore a different strategy is needed in the vicinity of such features.

In the practical implementation of the present method, two threshold values for the computed spatial distribution of spacing are used: a minimum spacing δ_{min} and a maximum spacing δ_{max}, so that

$$\delta_{min} \leq \delta_i \leq \delta_{max} \qquad \text{for} \quad i=1, \dots, N \quad (5.17)$$

The reason for defining the maximum value δ_{max} is to account for the possibility of a vanishing eigenvalue in (5.16) which would render that expression meaningless. The value of δ_{max} is chosen as the spacing which will be used in the regions where the flow is uniform (the far field, for instance). On the other hand, maximum values of the second derivatives occur near the discontinuities (if any) of the flow where the error indicator will demand that smaller elements are required. By imposing a minimum value δ_{min} for the grid size, we attempt to avoid an excessive concentration of elements near discontinuities. As the flow algorithm is known to spread discontinuities over a fixed number of elements (i.e. two or three), δ_{min} is therefore set to a value that is considered appropriate to ensure that discontinuities are represented to a required accuracy. This treatment also accounts for the presence of shocks of different strength in which, since the numerical values of the second derivative are different, equation (5.16) will assign them different grid spacings (e.g. larger spacings in the vicinity of weaker shocks).

The total number of elements generated in the new grid will now depend on the values selected for C, δ_{max}, and δ_{min}. However, it turns out that this number is mainly determined by the choice of the constant C, which is somehow arbitrary. The criterion employed here is to select a value that produces a computationally affordable number of elements.

The adaptive regriding strategy presented in this section is illustrated in figure 5.8 by showing the various stages during the adaptation process. Figure 5.8(a) shows the initial grid employed for the computation of the supersonic flow past a double ellipse configuration. The Mach number contours of the solution obtained on the initial grid are shown in figure 5.8(b). The flow conditions are a free stream Mach number of 8.15 and an angle of attack of 30^0. The application of expression 5.16 to the solution obtained produces the distribution of spacing and stretching displayed in figures 5.8(c) and 5.8(d) respectively. In figure 5.8(c) the contours corresponding to the value of the minimum spacing occuring in any direction is shown, whereas in figure 5.8(d) the value of and the direction of stretching is displayed in the form of a vector field. The magnitude of the vector represents the amount of stretching i.e. ratio between maximum and minimum spacings, and the direction of the vector indicates the direction along which the spacing is maximum. In this example expression 5.17 has been applied to the computed spacings with values of δ_{max} = 15 and δ_{min} = 0.9. Figures 5.8(e) - 5.8(h) show various stages during the regeneration process. It can be observed how small elements are generated first as discussed in section 3.5. The completed grid is shown in figure 5.8(i) and the solution computed on this adapted grid is shown in figure 5.8(j). It can be observed how a very significant improvement in the solution is obtained using, in this case, a single adaptation.

Estimating the Number of Elements to be Generated

The regeneration process uses the current grid as the background grid. Such a background grid clearly represents accurately the geometry of the computational domain. In this case, the number of elements to be generated, denoted by N_e, can be estimated as follows. Once the values of C, δ_{max}, and δ_{min} have been selected, the spatial distribution of grid parameters δ_i, α_i; i=1, ..., N is computed. For each element of the background grid, the values of the transformation **T** is computed at the centroid. The transformation is

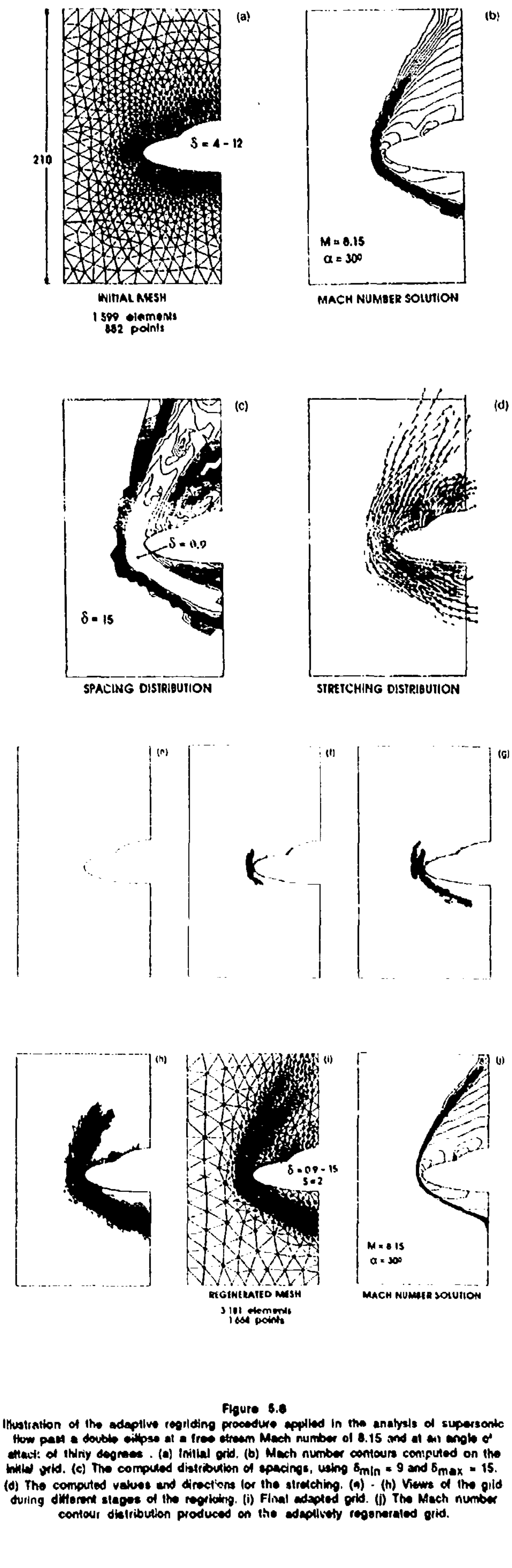


Figure 5.8
Illustration of the adaptive regriding procedure applied in the analysis of supersonic flow past a double ellipse at a free stream Mach number of 8.15 and at an angle of attack of thirty degrees . (a) Initial grid. (b) Mach number contours computed on the initial grid. (c) The computed distribution of spacings, using δ_{min} = 9 and δ_{max} = 15. (d) The computed values and directions for the stretching. (e) - (h) Views of the grid during different stages of the regriding. (i) Final adapted grid. (j) The Mach number contour distribution produced on the adaptively regenerated grid.

applied to the nodes of the element and its volume V_e in the normalised space is computed. The number of elements N_e is assumed to be proportional to the total volume in the unstretched space, i.e.

$$N_e \approx \chi \sum_{e=1}^{N_b} V_e \qquad (5.18)$$

where N_b is the number of elements in the background grid and χ is a constant. The value of χ is calculated as a statistical average of the values obtained for several generated grids. The calculated value is $\chi = 9$. This procedure gives estimates of the value of N_e with an error of less than 20%, which is accurate enough for most practical purposes. If the estimated value of N_e is either too big or too small, then the value of C is reduced or increased and the process repeated until the value of C produces a number of elements which is regarded as being computationally acceptable.

Application Examples

Double Ellipse

The adaptive regriding procedure is applied twice to the problem of flow past a double ellipse. The flow conditions are those previously considered for this configuration. The initial and two adapted grids and the solutions for Mach number are shown in figure 5.9. The characteristics of the grids employed are displayed in table 5.1.

Grid	Elements	Points	δ_{min}
1	2027	1110	4.0
2	3557	1864	0.9
3	6403	3294	0.25

Table 5.1 Double ellipse (M_∞=8.15, α=30^0): grid characteristics.

It is observed how the application of the adaptive procedure, when compared to the enrichment strategy, allows for a larger increase in the resolution at the expense of a smaller increase on total number of elements. On the other hand the regriding procedure does not suffer from the limitations inherent in the grid movement algorithm.

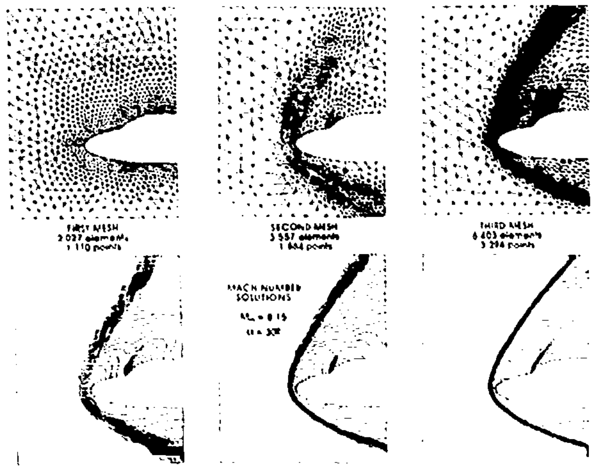


Figure 5.9
Supersonic flow past a double ellipse at a free stream Mach number of 8.15 and at an angle of attack of thirty degrees showing the sequence of grids and the corresponding solutions obtained using adaptive regriding

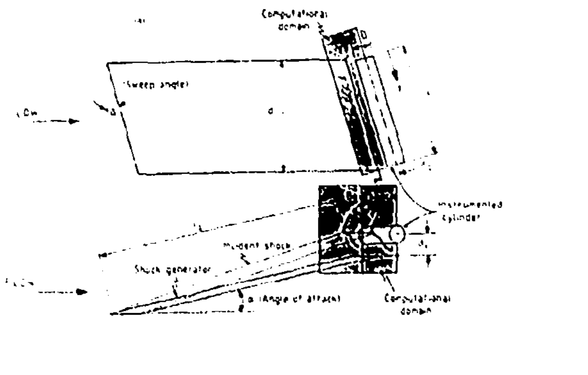
Shock Interaction on a Swept Cylinder

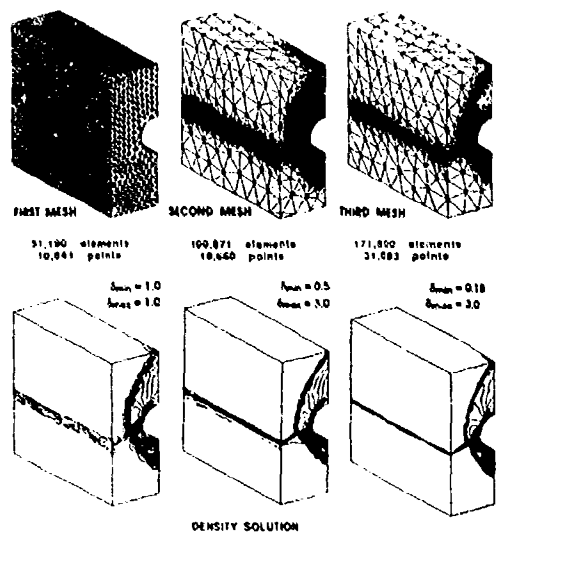
This is a problem of practical interest because its implications to the design of hypersonic vehicles [53]. The experimental apparatus and the computational domain adopted are shown diagramatically in figure 5.10(a). The numerical simulation has been carried out for a sweep angle of 15^0 on a cylinder of diameter D equal to 3 inches and length L equal to 9 inches. The undisturbed free stream Mach number is 8.03. The fluid which has been turned by the shock generator enters the computational domain with a Mach number of 5.26. The initial grid and those obtained after two adaptive regridings and the density contours distribution are shown in figure 5.10(b). The characteristics of the grids are shown in table 5.2.

Grid	Elements	Points	δ_{min}	δ_{max}
1	51 190	10 041	1.0	1.0
2	100 071	18 660	0.5	3.0
3	171 800	31 083	0.18	3.0

Table 5.2 3D Shock interaction on a swept cylinder: grid characteristics.

The potential advantages of the adaptive regriding procedure are clearly illustrated in this three dimensional example. The final adapted grid has a resolution of more than five times that of the initial grid whereas the total number of degrees of freedom increases by only a factor of 3.4. The effects of the three dimensional adaptation are best shown in figure 5.10(c) which shows the cross section through the grids half way along the cylinder. Two views of the three dimensional grid for the final adaptation together with the solution obtained are shown in figures 5.10(d) and 5.10(e) respectively.





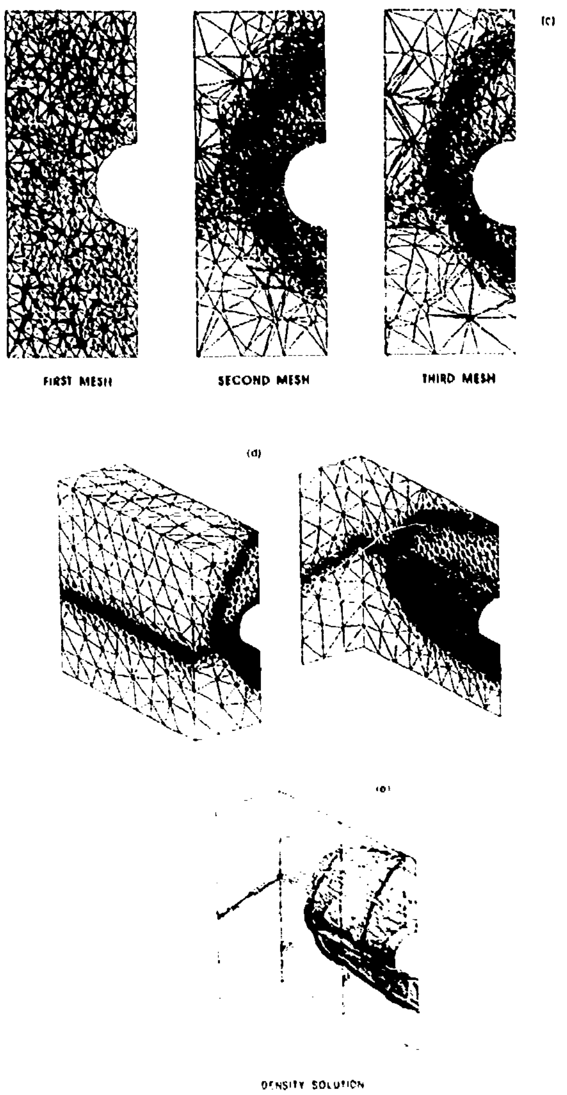


Figure 5.10
Shock interaction on a swept cylinder at a free stream Mach number of 8.15 and at fifteen degrees angle of sweep. (a) Sketch of the experimental apparatus and the chosen computational domain. (b) The sequence of grids and corresponding computed density contours obtained by using adaptive regriding. (c) Cross sections taken through the 3D grids half way along the cylinder. (d) The third adapted grid. (e) Detail of the shock interaction on the final grid.

Generic Fighter Configuration

This example concerns the simulation of the flow past a generic fighter configuration. The generation of the initial grid for that problem has been described in section 3.9. The flow conditions considered correspond to a free stream Mach number of 2 at an angle of attack of 3.79^0. The engine inlet is modelled by prescribing a Mach number of 0 3 within the engine. At the outlet supersonic flow conditions are assumed. Because of the symmetry of the problem only half of the domain is modelled. The spline definition of the geometry is shown in figure 5.11(a) and consists of 23 surface components and 53 curve components. Two grids have been employed in an initial demonstration of adaptive regriding applied to full aircraft configuration. The initial grid contains 76,522 tetrahedral elements with 4,128 triangular faces on the boundary. A preliminary first solution was computed using 1,500 iterations of the basic explicit scheme. A second grid was adaptively generated using the Mach number as the key variable in the error analysis. The new grid is formed by 70,125 tetrahedra with 7,262 triangles on

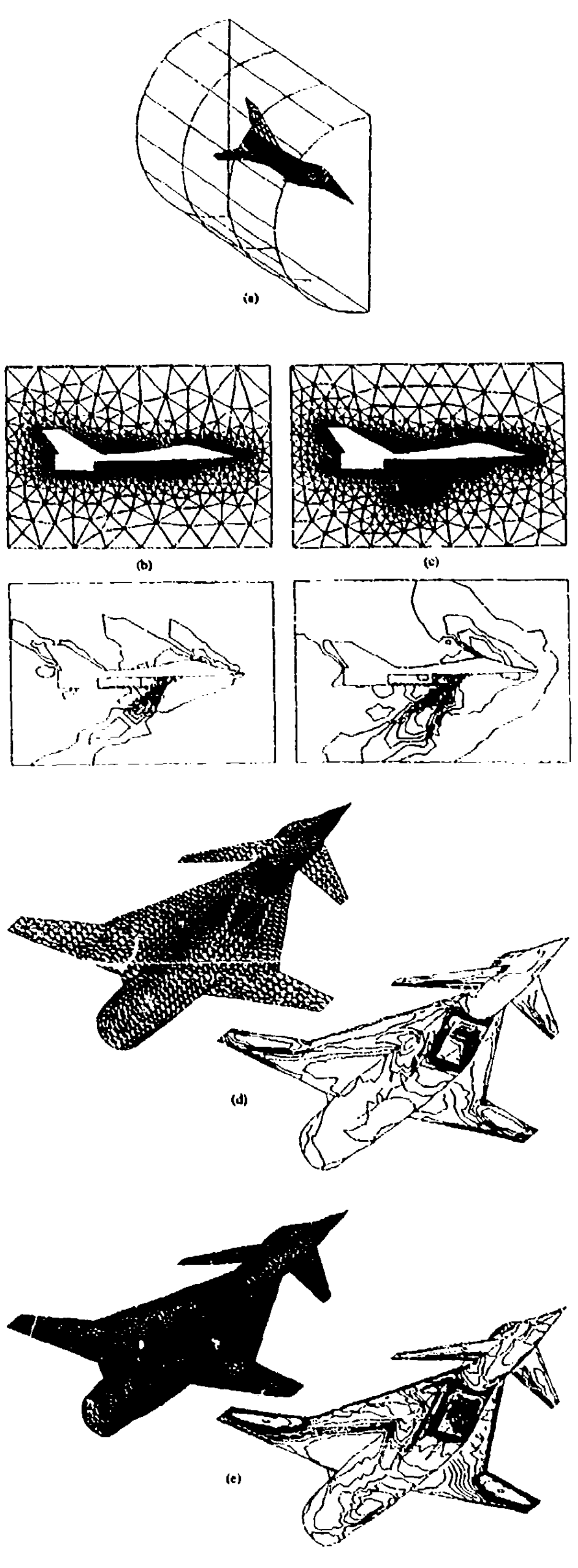


Figure 5.11
Generic fighter configuration at a free stream Mach number of 2 and at an angle of attack of 3.79 degrees. (a) Geometry definition - aircraft surface and outer boundary. (b) Initial grid and computed pressure solution in the symmetry plane. (c) Second grid and computed pressure solution on the symmetry plane. (d) Initial grid and computed pressure solution on the aeroplane surface. (e) Second grid and computed pressure solution on the aeroplane surface.

the boundary. It is interesting to notice that the number of elements in the two grids is approximately the same whereas the number of faces on the surface has increased. Moreover, the minimum spacing on the adapted grid is 3.5 times smaller than the one on the initial grid, thus indicating also an increase in the grid resolution. The solution on the new grid was obtained after 2,000 iterations. The grids and computed solutions at the plane of symmetry are shown in figures 5.11(b) for the initial grid and 5.11(c) for the adapted grid. The effect of the adaptation in the vicinity of the engine inlet can be observed. The grid and solution on the surface of the aircraft is shown in figure 5.11(d) for the initial grid and in figure 5.11(e) for the adapted grid. In this case the adaptation is very mild and is hardly noticeable. The main reason for this is that the resolution on the initial grid is rather poor and some important flow features are not properly captured.

6.TRANSIENT FLOWS [This section has been written in collaboration with E. J. Probert EBMS, University College, Swansea SA2 8PP, UK and O. Hassan, CDR, Innovation Centre, University College, Swansea SA2 8PP, UK]

6.1 Transient Flows

Solutions of the Euler equations are smooth over large areas of the computational domain and exhibit large gradients in localised parts of the flow. In transient simulations, these localised regions will generally move through the computational domain and may sweep across very large areas e.g. the case of flows involving propagating shocks. This means that, unless adaptivity is used, a globally fine grid will be necessary to provide the required resolution. Thus the use of adaptivity, with the possibility for local grid refinement and coarsening, offers the potential for considerable computational savings. We have already seen that only a few grid adaptations are generally needed to obtain a satisfactory solution to a steady problem, but we can expect that grid adaptation will have to be performed several thousand times in a transient flow simulation. Thus any potential computational savings which appear to be offered by the use of adaptivity in this case will only be realised if the adaptation of the grid can be performed in an efficient manner. Successful implementations of adaptivity to the solution of transient problems have already been made within the context of both structured [54] and unstructured grids [55,56].

6.2 Grid Enrichment

An obvious method of achieving grid adaptation for transient flow simulation is the extension of the grid enrichment ideas introduced above for the solution of steady state problems. An extremely successful implementation on unstructured triangular grids has been made by Löhner [55]. In his method, the grid is automatically refined and de-refined as necessary according to the results of an error indicating process. An example [56] of the application of this procedure to shock-bubble interaction problem is shown in figure 6.1. This problem involves the interaction between a weak shock, travelling at a Mach number of 1.29 in air, and a bubble of heavier material (freon). From the figure it can be seen how the shock speed inside the bubble decreases, owing to the higher density of the freon, whereas the outer shock bends over. The inner shock focuses at the right hand end of the bubble, producing a significant over-pressure and initiating a small circular blast wave. This method has also recently been applied to three dimensional flow simulations [57].

6.3 Transient Flows Involving Moving Bodies

The complexity involved in transient flow simulation increases if one considers problems in which certain boundaries of the computational domain are allowed to move, so that the geometry of the domain changes with time. This means that the grid must be modified during the computation in order to accommodate these geometrical changes. One approach which has proved to be successful for tackling such problems is the chimera approach, in which each individual geometry component can have its own associated structured grid which can move independently of the other grids. Three dimensional viscous simulations involving moving bodies have already been produced by this method [58]. Unstructured grids have been applied to the solution of inviscid two dimensional transient flows involving moving bodies [59,60], using a method which is an extension of the regriding procedures presented in section 5.6 and this is the approach that will be presented here.

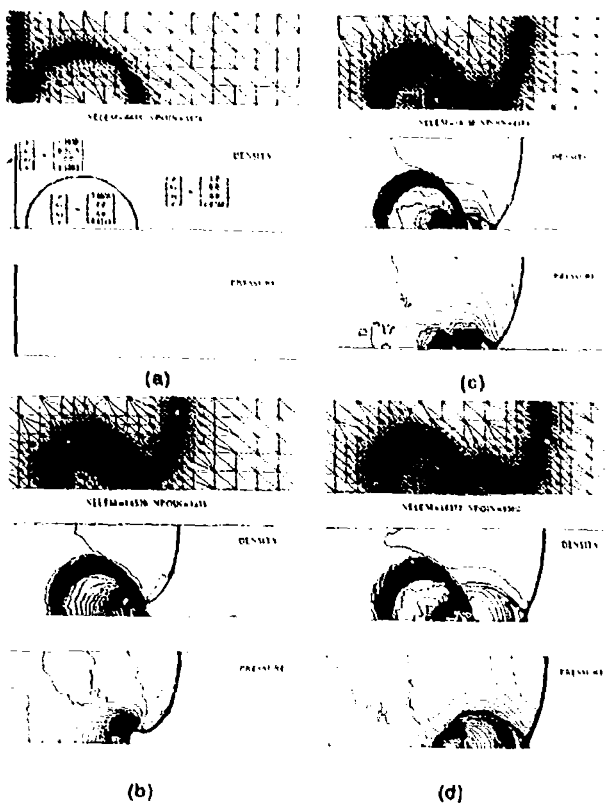


Figure 6.1
Shock bubble interaction problem using transient adaptation based on enrichment (from [56]). (a) Initial grid and solution contours. (b) Grid and solution at t=0.6 (c) Grid and solution at t=0.7 (d) Grid and solution at t=0.8

We restrict our consideration to two dimensional inviscid flows and note that the basic variational statement for the problem will need to be modified to account for the fact that the spatial domain Ω is varying with time. Suppose that we have the solution $\underline{U}_n$ at a certain time level t_n. We attempt to satisfy the compressible Euler equations (1.38) over the space-time domain $D = \{\Omega(t), t_n \le t \le t_{n+1}\}$. To express this problem in a variational form we need to introduce suitable trial and weighting function sets. We assume, for the purposes of this discussion, that the conditions on the boundary Γ of Ω can be expressed in the form

$$\underline{U} = \underline{0} \qquad \text{on } \Gamma(t) \tag{6.1}$$

Although such conditions are somewhat unrealistic, the actual boundary conditions which would need to be applied in the simulation of a given problem can be readily incorporated by making appropriate modifications to the following analysis. We may define

$$\mathcal{T} = \{ \underline{U} \mid \underline{U} = \underline{0} \text{ on } \Gamma; \underline{U} = \underline{U}_n \text{ on } \Omega \text{ at } t = t_n \}$$

$$\mathcal{W} = \{ W \mid W = 0 \text{ on } \Gamma \} \tag{6.2}$$

and a variational formulation of the problem can be stated as : find $\underline{U}$ in $\mathcal{T}$ such that

$$\int_{t_n}^{t_{n+1}} \int_{\Omega} W \left[\frac{\partial \underline{U}}{\partial t} + \frac{\partial \underline{E}}{\partial x} + \frac{\partial \underline{F}}{\partial y} \right] d\Omega \, dt = \underline{0} \tag{6.3}$$

for every W in $\mathcal{W}$. We will assume that the spatial domain Ω has been discretised using 3 noded linear triangular elements, with interior nodes numbered from 1 to p and introduce the sets

$$\mathcal{T}_{(p)} = \{ \underline{U}_{(p)} \mid \underline{U}_{(p)} = M_1 \underline{U}_1 + M_2 \underline{U}_2 + \ldots + M_p \underline{U}_p ;\ \underline{U}_{(p)} = \underline{0} \text{ on } \Gamma;\ \underline{U}_{(p)} = \underline{U}_n \text{ on } \Omega \text{ at } t = t_n \}$$

$$\mathcal{W}_{(p)} = \{ W_{(p)} \mid W_{(p)} = a_1 M_1 + a_2 M_2 + \ldots + a_p M_p ;\ W_{(p)} = 0 \text{ on } \Gamma \} \qquad (6.4)$$

In the approach which is to be followed here, certain nodes in the grid will be fixed, while others will move with a prescribed velocity. The shape functions M_J are linear functions of space and time which satisfy

$$M_J(\underline{x}, t_n) = N_J^n(\underline{x}) \qquad M_J(\underline{x}, t_{n+1}) = N_J^{n+1}(\underline{x}) \qquad (6.5)$$

where N_J^n is the standard finite element shape function, defined in section 1.2, associated to node J at time t_n. Working with the function sets defined in equation (6.4), the Galerkin approximation statement takes the form : find $\underline{U}_{(p)}$ in $\mathcal{T}_{(p)}$ such that

$$\int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[\frac{\partial \underline{U}_{(p)}}{\partial t} + \frac{\partial \underline{F}_{(p)}}{\partial x} + \frac{\partial \underline{G}_{(p)}}{\partial y} \right] d\Omega \, dt = \underline{0} \qquad (6.6)$$

for J = 1,2,... ,p. Considering the first term appearing in this integral, it is possible to show that

$$\int_{\Omega} M_J \frac{\partial \underline{U}_{(p)}}{\partial t} d\Omega = \frac{d}{dt} \int_{\Omega} M_J \underline{U}_{(p)} d\Omega - \int_{\Gamma} \underline{v} \cdot \underline{n} \, M_J \underline{U}_{(p)} d\Gamma - \int_{\Omega} \frac{\partial M_J}{\partial t} \underline{U}_{(p)} d\Omega \qquad (6.7)$$

where $\underline{v}$ denotes the velocity of the moving nodes. With $\underline{v}_{(p)} = (v_x, v_y)$ interpolated linearly between the nodal values of $\underline{v}$, an observer moving with the grid will not detect any change in the shape functions i.e.

$$\frac{DM_J}{Dt} = \frac{\partial M_J}{\partial t} + v_x \frac{\partial M_J}{\partial x} + v_y \frac{\partial M_J}{\partial y} = 0 \qquad (6.8)$$

where D/Dt denotes differentiation following the moving grid and so

$$\int_{\Gamma} \underline{v} \cdot \underline{n} \, M_J \underline{U}_{(p)} d\Gamma + \int_{\Omega} \frac{\partial M_J}{\partial t} \underline{U}_{(p)} d\Omega = \int_{\Omega} M_J \left[\frac{\partial v_x \underline{U}_{(p)}}{\partial x} + \frac{\partial v_y \underline{U}_{(p)}}{\partial y} \right] d\Omega \qquad (6.9)$$

Finally, combining equations (6.6), (6.7) and (6.9), the Galerkin approximation satisfies

$$\int_{\Omega_{n+1}} M_J \underline{U}_{(p)} d\Omega - \int_{\Omega_n} M_J \underline{U}_{(p)} d\Omega = - \int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[\frac{\partial \underline{F}_{(p)}^*}{\partial x} + \frac{\partial \underline{G}_{(p)}^*}{\partial y} \right] d\Omega \, dt \qquad (6.10)$$

where

$$\underline{F}_{(p)}^* = \underline{F}_{(p)} - v_x \underline{U}_{(p)} \qquad \underline{G}_{(p)}^* = \underline{G}_{(p)} - v_y \underline{U}_{(p)} \qquad (6.11)$$

Inserting the assumed form for $\underline{U}_{(p)}$ from equation (6.4),

$$[\mathbf{M}\,\underline{U}]_J^{n+1} - [\mathbf{M}\,\underline{U}]_J^{n} = - \int_{t_n}^{t_{n+1}} \int_{\Omega} M_J \left[\frac{\partial \underline{F}_{(p)}^*}{\partial x} + \frac{\partial \underline{G}_{(p)}^*}{\partial y} \right] d\Omega \, dt \qquad (6.12)$$

The integral appearing here can be evaluated by first employing one point integration in time (at $t = t_{n+1/2}$) and then using a two-step approximation [18,19]. Artificial viscosity will again be needed with a scheme of this type and the resolution of the resulting scheme may be improved by the use of the FCT ideas mentioned in Section 1.3.

6.4 Adaptive Regridding for Transient Flows Involving Moving Bodies

The method described above, whereby a grid may be adapted by regriding, is a natural approach to follow for the simulation of flows involving moving bodies. It will be assumed that the motion of any moving boundary is prescribed and the objective is to determine the resulting flow field. The description of an algorithm which can be devised to advance the solution of equation (6.11) in time can be written as follows:

1. Generate an initial grid to represent the computational domain and to adequately resolve the initial solution.
2. Start the time-step loop
2.1 Advance the solution one time-step
2.2 Update the coordinates of the points on the moving boundaries
2.3 Use an error indicator to examine the current solution and define an 'optimal' distribution of grid spacing and stretching
2.4 Compare the current grid with the 'optimal' grid. Delete the elements whose size and shape is too different from the optimal
2.5 Triangulate the regions where elements have been deleted according to the new distribution of grid parameters
2.6 Determine, by interpolation, the flow variables at the new nodes
End the time-step loop

It is apparent that the crucial phase in this process is the grid adaptation in steps 2.3-2.5. The mechanics of this process is illustrated diagramatically in figure 6.2. The success of the procedure depends upon the reliability of the error indicator which is employed. The indicator of equation (5.9) has again been used for this application.

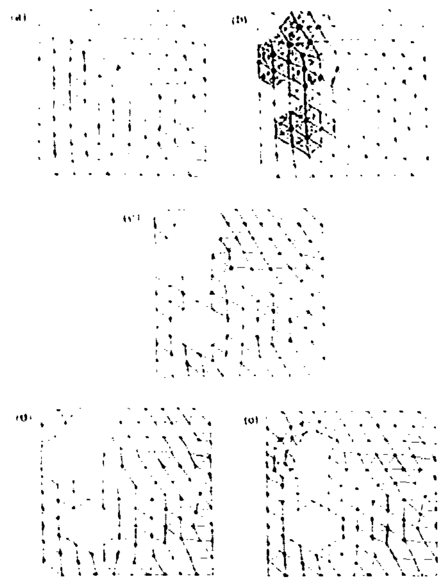


Figure 6.2
Illustration of the adaptive remeshing procedure applied to transient problems (a) Initial grid (b) Marked unwanted nodes and elements. (c) Elements are removed from the grid (d) Boundary sides are generated according to the new spacing distribution to foram a closed loop arround each hole (e) Triangulation of the holes using the advancing front and the new distribution of spacings.

6.5 Application Examples

1D Shock Propagation

The first example considered consists of a two dimensional simulation of the transient development of the flow in shock tube. The purpose of this example is to illustrate the application of the regriding algorithm to a transient problem with fixed boundaries. The initial conditions are such that the solution consists of a single propagating shock. Figure 6.3 depicts the development of the grid and solution as the shock propagates. It can be observed that the shock movement is adequately followed by the adaptation of the grid. Note also that the number of elements in the grid remains approximately constant as the solution progresses.

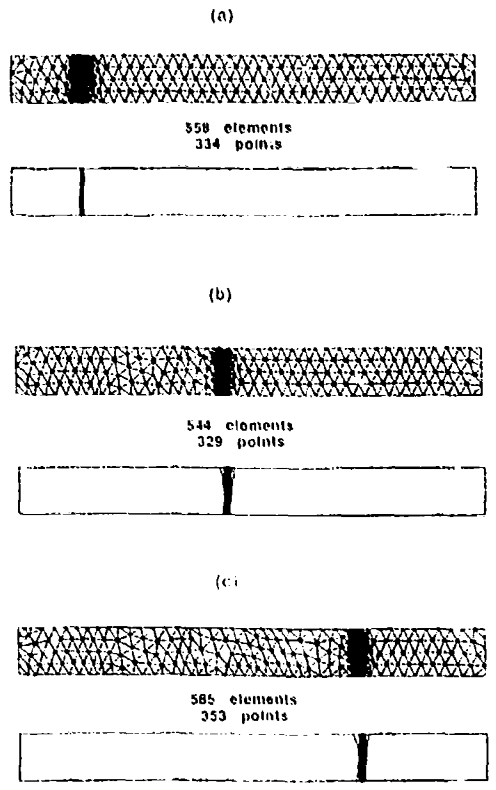


Figure 6.3
Shock tube example using adaptively refined meshes (a) Adapted grid and density contours at t=0 (b) Adapted grid and density contours at t=4 (c) Adapted grid and density contours at t=8

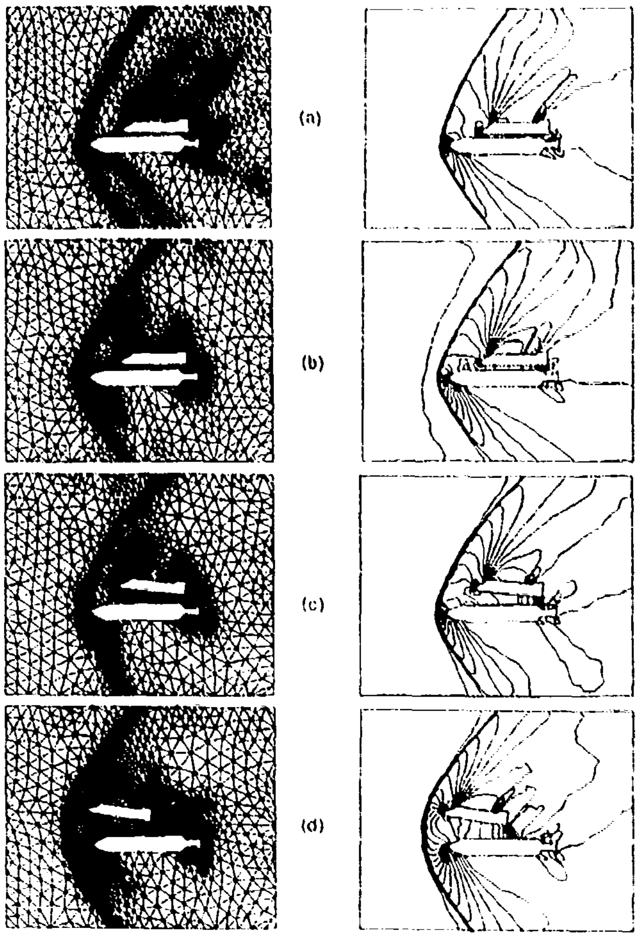


Figure 6.1
Space shuttle vehicle and booster rocket simulation at a free stream Mach number of 2 and at an angle of attack of -4 degrees. The shuttle moves upwards and away from the rocket with a prescribed motion. Sequence of meshes and pressure solutions obtained during the transient simulation starting from a steady state solution. The grids consist of (a) 7,870 elements and 4,130 points, (b) 7,377 elements and 3,867 points and (d) 8,459 elements and 4,379 points

Integrated Space Shuttle Vehicle Simulation

A two dimensional computation has been attempted which involves a simulated space shuttle separating from the rocket booster. The relative motion of the shuttle with respect to the booster has been prescribed externally. The free stream conditions correspond to a Mach number of 2 and an angle of attack of -4^0 with respect to the initial position of the shuttle. An initial steady state was computed for the configuration shown in figure 6.4(a). Figures 6.4(b)-6.4(d) illustrate the development of the grid and solution as the separation proceeds.

ACKNOWLEDGEMENTS

Our work in this area has benefited greatly from the support over several years of the Aerothermal Loads Branch, NASA Langley Research Center, under grants NAGW-478 and NAGW-1809 and we are pleased to acknowledge this contribution. We also acknowledge the assistance of Dassault Aviation for partially funding this work and providing the geometry definition of the Falcon; NASA Ames for the support provided under grant NCCW-008 and Rolls-Royce for providing the geometry definition and the support which allowed us to perform the computation of the flow past the installed nacelle.

REFERENCES

1. N.P. Weatherill, 'Mesh generation in computational fluid dynamics', *von Karman Insitute for Fluid Dynamics*, Lecture Series 1989-04, Brussels, 1989.

2. J.F. Thompson, Z.U.A. Warsi and C.W. Mastin, 'Numerical grid generation - foundations and applications', *North-Holland*, 1985.

3. S. Allwright, 'Multiblock topology specification and grid generation for complete aircraft configurations', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 11.1-11.11, 1990.

4. T.J. Baker, 'Unstructured mesh generation by a generalized Delaunay algorithm', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 20.1-20.10, 1990.

5. J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comp. Phys. 72*, 449-466, 1987.

6. A. Jameson, T.J. Baker and N.P. Weatherill, 'Calculation of inviscid transonic flow over a complete aircraft', *AIAA Paper 86-0102*, 1986.

7. J. Peraire, J. Peiró, L. Formaggia, K. Morgan and O.C. Zienkiewicz, 'Finite element Euler computations in three dimensions', *Int. J. Num. Meth. in Engn.*, 26, 1988.

8. R. Löhner, K. Morgan and O.C. Zienkiewicz, 'Adaptive grid refinement for the compressible Euler equations', in *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, Edited by I. Babuska et al, Wiley, 281-297, 1986

9. L. Formaggia, J. Peraire, K. Morgan and J. Peiró, 'Implementation of a 3D explicit Euler solver on a CRAY computer', in *Proc. 4th Int. Symposium on Science and Engineering on CRAY Supercomputers*, Minneapolis, 45-65, 1988.

10. C. Hirsch, 'Numerical computation of internal and external flows, Volume 2', *Wiley*, 1990.

11. C. Johnson, 'Numerical solutions of partial differential equations by the finite element method', *Cambridge University Press*, 1987.

12. O.C. Zienkiewicz and K. Morgan, 'Finite Elements and Approximation', *Wiley*, 1983.

13. J. Donéa, 'A Taylor-Galerkin method for convective transport problems', *Int. J. Num. Meth. Engng. 20*, 101-119, 1984.

14. O. Hassan, K. Morgan and J. Peraire, 'An adaptive implicit/explicit finite element scheme for compressible viscous high speed flows', *AIAA Paper 89-0363*, 1989.

15. K. Morgan and J. Peraire, 'Finite element methods for compressible flows', *von Karman Institute for Fluid Dynamics, Lecture Series 1987-04*, Brussels 1987.

16. J. Donéa and S. Giuliani, 'A simple method to generate high-order accurate convection operators for explicit schemes based on linear finite elements', *Int. J. Num. Meth. Fluids 1*, 63-79, 1981.

17. K. Morgan and J. Peraire, 'An introduction to finite element methods for computational fluid dynamics', *Institute for Numerical Methods in Engineering Report C/R/608/88*, University College of Swansea 1988.

18. J. Peraire, K. Morgan and J. Peiró, 'Unstructured finite element mesh generation and adaptive procedures for CFD', in *Applications of Mesh Generation to Complex 3-D Configurations*, AGARD Conference Proceedings No. 464, 18.1-18.12, 1990.

19. R. Löhner, K. Morgan, J. Peraire and O.C. Zienkiewicz, 'Finite element methods for high speed flows', *AIAA Paper 85-1531-CP*, 1985

20. J.P. Boris and D.L. Book, 'Flux corrected transport I : SHASTA, a fluid transport algorithm that works', *J. Comp. Phys. 11*, 8-69, 1973.

21. S. T. Zalesak, 'Fully multidimensional flux corrected transport algorithm for fluids', *J. Comp. Phys. 31*, 335-362, 1979.

22. A.K. Parrott and M.K. Christie, 'FCT applied to the 2D finite element solution of tracer transport by single phase flow in a porous medium', in *Numerical Methods for Fluid Dynamics*, Edited by K.W. Morton and M.J. Baines, Oxford University Press, 27-53, 1986.

23. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux corrected transport (FEM-FCT) for the Euler and Navier-Stokes equations', in *Finite Elements in Fluids, Volume 7*, Edited by R.H. Gallagher et al, 105-121, 1988.

24. K. Morgan, J. Peraire and R. Löhner, 'Adaptive finite element flux corrected transport techniques for CFD', in *Finite Elements - Theory and Application*, Edited by D.L. Dwoyer et al., Springer-Verlag, 165-175, 1988.

25. A. Lerat and J. Sides, 'Efficient solution of the steady Euler equations with a centered implicit scheme', in *Numerical Methods for Fluid Dynamics III*, K.W Morton and M.J. Baines ed., 65-86, Clarendon Press, Oxford, 1988.

26. O. Hassan, K. Morgan and J. Peraire, 'An implicit finite element method for high speed flows', *AIAA Paper-90-0402*, 1990

27. A.A.G. Requicha and H.B. Voelcher,' Solid modelling. A Historical Summary and contemporary assessment', *IEEE Computer Graphics and Applications*, 3, n. 2, 9-24, 1982.

28. I.D. Faux and M.J. Pratt, Computational Geometry for Design and Manufacture, *Ellis Horwood*, Chichester, 1981.

29. J.C. Ferguson, 'Multivariate curve interpolation', *J. ACM 11*, 221 ,1964.

30. S.A. Coons, 'Surfaces for Computed Aided Design of Space Forms', *Report MAC-TR-41, Project MAC*, M.I.T., 1967

31. J. Peiró,'A finite element procedure for the solution of the Euler equations using unstructured meshes', *University of Wales Ph.D. Thesis*, C/Ph/126/89, 1989.

32. A.J. George, 'Computer implementation of the finite element method', *Ph. D Thesis, Stanford University*, STAN-CS-71-208, 1971.

33. S.H. Lo, 'A new mesh generation scheme for arbitrary planar domains', *Int. J. Numer. Methods Engng.* 21, 1403-1426 (1985).

34. J.C. Cavendish, D.A. Field and W.H. Frey, 'An approach to automatic three dimensional finite element mesh generation', *Int. J. Num. Meth. Engng* 21, 329-348,1985

35. J.J. Stoker, Differential geometry, *Wiley Interscience*, New York, 1969

36. L.-E. Eriksson, R.E. Smith, M.R. Wiese and N. Farr, 'Grid generation and inviscid flow computation about cranked-winged airplane geometries', *AIAA Paper 87-1125*,1987.

37. J.L. Bentley and J.H. Friedman, 'Data structures for range searching', *Computing Surveys, 11*, 4, 1979.

38. M.I. Shamos and D. Hoey, 'Geometric intersection problems', In *17th Annual Symposium on Foundations of Computer Science*, IEEE, 1976.

39. 'Fundamental algorithms for computer graphics', edited by R.A. Earnshaw, *NATO ASI Series F, vol. 17*, Springer Verlag 1985.

40. J.Boris, 'A vectorised algorithm for determining the nearest neighbours', *J. Comp. Phys., 66*, 1-20, 1986.

41. R. Sedgewick, 'Algorithms', *Addison Wesley*, Reading, 1988

42. J. Bonet and J. Peraire, "An alternating digital tree (ADT) algorithm for geometric searching and intersection problems", *Int. J. Num. Meth. Engng.*, in press (1990).

43. D. Knuth, 'The art of computer programming - Sorting and searching', vol. 3, *Addison Wesley*, 1973

44. D. Knuth, 'The art of computer programming - Fundamental algorithms', vol. 1, *Addison Wesley*, 1969.

45. J.L. Bentley, ' Multidimensional binary search trees used for associative searching', *Communications of the ACM, 18*, 1, 1975.

46. J. Bonet, 'Finite element analysis of thin sheet superplastic forming processes', *University of Wales Ph.D. Thesis*, C/PhD/128/89.

47. P.G. Ciarlet and P.A. Raviart, 'General Lagrange and Hermite interpolation in R^n with applications to finite element methods', *Arch. Rat. Mech. Anal., 46*, 177-199, 1972.

48. J.T. Oden Grid optimisation and adaptive meshes for finite element methods, *University of Texas at Austin Notes*, 1983.

49. B. Palmerio, V. Billey, A. Dervieux and J. Periaux, 'Self-adaptive mesh refinements and finite element methods for solving the Euler equations', In *Numerical Methods for Fluid Dynamics II*, K.W Morton and M.J. Baines ed., 369-388, Clarendon Press, Oxford, 1985.

50. J. Peraire, K. Morgan, J. Peiró and O.C. Zienkiewicz, 'An adaptive finite element method for high speed flows', *AIAA Paper 87-0556*, 1987.

51. B. Palmerio and A. Dervieux, '2D and 3D unstructured mesh adaption relying on physical analogy', in Proc. of the *Second International Conference on Numerical Grid Generation in Computational Fluid Mechanics*, Miami Beach, Florida, 1988.

52. K. Nakahashi and G.S. Deiwert, 'A practical adaptive-grid method for complex fluid flow problems', *Lecture Notes in Physics, vol 218*, 422-426, Springer Verlag, 1985.

53. C. Glass, A.R. Wieting and M. Holden, 'Effect of leading edge sweep on shock-shock interference heating at Mach 8', *AIAA Paper 89-0271*, 1989

54. P. Woodward and P. Colella, 'The numerical simulation of two dimensional fluid flow with strong shocks', *J. Comp Phys. 54*, 115-173 , 1984.

55. R. Löhner, 'An adaptive finite element scheme for transient problems in CFD' *Comp. Meth. in Appl. Mech and Engn. 61*, 323-338, 1987.

56. R. Löhner, K. Morgan, J. Peraire and M. Vahdati, 'Finite element flux corrected transport for the Euler and Navier-Stokes equations', *Int. J. Num. Meth. in Fluids 7*, 1093-1109, 1987.

57. R. Löhner and J.D. Baum, 'Numerical simulation of shock interaction with complex geometry three-dimensional structures using a new adaptive h-refinement scheme on unstructured grids', *AIAA Paper 90-0700*, 1990.

58. P.G. Buning, I.T. Chiu, S. Obayashi, Y.M. Rizk and J.L. Steger, 'Numerical simulation of the integrated space shuttle vehicle in ascent', *AIAA Paper 88-4359-CP*, 1988

59. L. Formaggia, J. Peraire and K. Morgan, 'Simulation of a store separation using the finite element method', *Appl. Math. Modelling* 12, 175-181, 1988.

60. E.J. Probert, 'Finite element methods for transients compressible flows', *University of Wales Ph.D Thesis*, C/Ph/123/89, 1989.

61. T.J. Barth, 'Numerical aspects of computing viscous high Reynolds number flows on unstructured meshes', *AIAA Paper 91-0721*, 1991

62. D.J. Mavriplis, 'Three dimensional unstructured multigrid for the Euler equations', *AIAA Paper 91-1549*, 1991

63. P. Roe, 'Approximate Riemann solvers, parameter vectors and difference schemes', *J.Comp. Phys.* 43, 357-372, 1981

64. A. Jameson, 'Transonic flow calculations', *Princeton University Report MAE 1751*, 1984

65. A. Jameson and W. Schmidt, 'Some recent developments in numerical methods for transonic flows', *Comp.Meth.Appl.Mech. Engng.* 51, 467-493, 1985

66. K.Morgan, J.Peraire, R.R. Thareja and J.R. Stewart, 'An adaptive finite element scheme for the Euler and Navier-Stokes equations', *AIAA 8th Computational Fluid Dynamics Conference*, Honolulu, Hawaii, 1987

67. R.C. Swanson and E. Turkel, 'On central-difference and upwind schemes', *NASA Langley Research Center ICASE Report No.* 90-44, 1990

68. M. Giles, 'Energy stability analysis of multi-step methods on unstructured meshes', *M.I.T. CFD Laboratory Report* CFDL-TR-87-1, 1987

69. J. Peiró, J. Peraire, K. Morgan, O. Hassan and N. Birch, 'The numerical simulation of flow about installed aero-engine nacelles using a finite element solver on unstructured meshes', accepted for publication in *Aero. J.*, 1992

70. E. Perez, 'Finite Element and multigrid solution of the two-dimensional Euler Equations on a non-structured mesh', *INRIA Report* No.442, 1985

71. R. Löhner and K. Morgan, 'Unstructured multigrid methods for elliptic problems', *Int.J.Num.Meth.Engng.* 24, 101-115, 1987

72. D.J. Mavriplis, 'Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes', *AIAA J.* 26, 824-831, 1988

73. M.-P. Leclercq, J. Periaux and B. Stoufflet, 'Multigrid methods with unstructured methods', *Proc. 7th Int. Conf. on Finite Elements in Flow Problems*, Huntsville, Alabama, 1989

ASPECTS OF UNSTRUCTURED GRIDS AND FINITE-VOLUME SOLVERS FOR THE EULER AND NAVIER-STOKES EQUATIONS

by

Timothy J. Barth
CFD Branch
NASA Ames Research Center
Moffett Field, CA 94035
United States

Contents

Introduction

One of the major achievements in engineering science has been the development of computer algorithms for solving nonlinear differential equations such as the Navier-Stokes equations. These algorithms are now used in the practical engineering design of devices such as cars and airplanes as well as theoretical studies of complex phenomena such as fluid turbulence. In past years, limited computer resources have motivated the development of efficient numerical methods in computational fluid dynamics (CFD) utilizing *structured* meshes. These meshes are comprised of systematic arrays of quadrilateral or hexahedral cells. The use of structured meshes greatly simplifies the implementation of CFD algorithms on conventional computers. Structured meshes also permit the use of highly efficient solution techniques such as alternating direction implicit (ADI) iteration schemes or multigrid. Following the dramatic improvement in computing speed in recent years, emphasis has shifted towards the design of algorithms capable of treating complex geometries. The automatic generation of structured grids about complex geometries is problematic. Unstructured grids offer one promising alternative technique for treating these general geometries. Unstructured meshes have irregular connectivity and usually contain triangles and/or quadrilaterals in two dimensions and tetrahedra and/or hexahedra in three dimensions. The generation and use of unstructured grids poses new challenges in computational fluid dynamics. This is true for both grid generation as well as for the design of algorithms for flow solution. The purpose of these notes is to present recent developments in the unstructured grid generation and flow solution technology.

1.0 Preliminaries

1.1 Graphs and Meshes

Graph theory offers many valuable theoretical results which directly impact the design of efficient algorithms using unstructured grids. For purposes of the present discussion, only *simple* graphs which do not contain self loops or parallel edges will be considered. Results concerning simple graphs usually translate directly into results relevant to unstructured grids. The most famous graph theoretic result is Euler's formula which relates the number of edges $n(e)$, vertices $n(v)$, and faces $n(f)$ of a polyhedron (see figure 1.0(a)):

$$n(f) = n(e) - n(v) + 2 \qquad \text{(Euler's formula)} \qquad (1.0)$$

This polyhedron can be embedded in a plane by mapping one face to infinity. This makes the graph formula (1.0) applicable to 2-D unstructured meshes. In the example below, the face 1-2-3-4 has mapped to infinity to form the exterior (infinite) face. If all faces are numbered including the exterior face, then Euler's formula (1.0) remains valid.

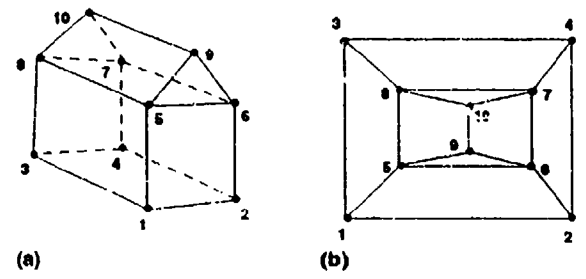


Figure 1.0 (a) 3-D Polyhedron, (b) 2-D Planar embedding

The infinite face can be eliminated by describing the outer boundary in terms of boundary edges which share exactly one interior face (interior edges share two). We also consider boundary edges which form simple closed curves in the interior of the mesh. These curves serve to describe possible objects embedded in the mesh (in this case, the polygon which they form is not counted as a face of the mesh). The number of these polygons is denoted by $n(h)$. The modified Euler's formula now reads

$$n(f) + n(v) = n(e) + 1 - n(h) \qquad (1.1)$$

Since interior edges share two faces and boundary edges share one face, the number of interior and boundary edges can be related to the number of faces by the following formula:

$$2n(e)_{interior} + n(e)_{bound} = \sum_{i=3}^{max\ d(f)} i\ n(f)_i \quad (1.2)$$

where $n(f)_i$ denotes the number of faces of a particular edge degree, $d(f) = i$. Note that for pure triangulations T, these formulas can be used to determine, *independent of the method of triangulation*, the number of triangles or edges given the number of vertices $n(v)$, boundary edges $n(e)_{bound}$, and interior holes $n(h)$

$$n(f)_3 = 2n(v) - n(e)_{bound} - 2 + 2n(h) \quad (1.3)$$

or

$$n(e) = 3n(v) - n(e)_{bound} - 3 + 3n(h). \quad (1.4)$$

This is a well known result for planar triangulations. (For brevity, we will sometimes use N to denote $n(v)$ in the remainder of these notes.) In many cases boundary edges are not explicitly given and the boundary is taken to be the convex hull of the vertices, i.e. the smallest convex polygon surrounding the vertices. (To obtain the convex hull in two dimensions, envision placing an elastic rubber band around the cloud of points. The final shape of this rubber band will be the convex hull.) ***A key observation for planar meshes is the asymptotic linear storage requirement with respect to the number of vertices for arbitrary mesh arrangements.***

The Euler formula extends naturally to an arbitrary number of space dimensions. In this general setting, Euler's formula relates basic components (vertices, edges, faces, etc.) of higher dimensional polytopes. In computational geometry jargon, vertices, edges, and faces are all specific examples of "k-faces". A 0-face is simply a vertex, a 1-face corresponds to a edge, a 2-face corresponds to a *facet* (or simply a face), etc. A polytope $\mathcal{P}$ in $\mathbf{R}^d$ contains k-faces $\forall\ k \in \{-1, 0, 1, ..., d\}$. The -1-face denotes the *null set* face by standard convention. Let the number of k-faces contained in the polytope $\mathcal{P}$ be denoted by $N_k(\mathcal{P})$. For example, $N_0(\mathcal{P})$ would denote the number of vertices. Using this notation, we have the following relationships:

$$\begin{array}{ll} N_0 \equiv n(v) \text{ (vertices)}, & N_1 \equiv n(e) \text{ (edges)} \\ N_2 \equiv n(f) \text{ (faces)}, & N_3 \equiv n(\phi) \text{ (volumes)} \end{array}$$

By convention, there is exactly one null set contained in any polytope, i.e. $N_{-1}(\mathcal{P}) = 1$ and by definition $N_d(\mathcal{P}) = 1$. Using these results, we can succinctly state the general Euler formula for an arbitrary polytope in $\mathbf{R}^d$

$$\sum_{k=-1}^{d} (-1)^k N_k(\mathcal{P}) = 0 \quad \text{(Euler's Formula in } \mathbf{R}^d) \tag{1.5}$$

On the surface of a polyhedron in 3-space, the standard Euler formula (1.0) is recovered since

$$-1 + N_0 - N_1 + N_2 - 1 = 0$$

or

$$n(f) + n(v) = n(e) + 2.$$

To obtain results relevant to three-dimensional unstructured grids, the Euler formula (1.5) is applied to a four-dimensional polytope.

$$n(f) + n(v) = n(e) + n(\phi) \tag{1.6}$$

This formula relates the number of vertices, edges, faces, and volumes $n(\phi)$ of a three-dimensional mesh. As in the two-dimensional case, this formula does not account for boundary effects because it is derived by looking at a single four-dimensional polytope. The example below demonstrates how to derive exact formulas including boundary terms for a tetrahedral mesh. Derivations valid for more general meshes in three dimensions are also possible.

Example: Derivation of Exact Euler Formula for 3-D Tetrahedral Mesh.

Consider the collection of volumes incident to a single vertex v_i and the polyhedron which describes the shell formed by these vertices. Let $F_b(v_i)$ and $N_b(v_i)$ denote the number of faces and vertices respectively of this polyhedron which actually lie on the boundary of the entire mesh. Also let $E(v_i)$ denote the total number of edges on the polyhedron surrounding v_i. Finally, let $d_\phi(v_i)$ and $d_e(v_i)$ denote the number of tetrahedral volumes and edges respectively that are incident to v_i. On this polyhedron, we have exact satisfaction of Euler's formula (1.0), i.e.

$$\overbrace{d_\phi(v_i) + F_b(v_i)}^{\text{polyhedral faces}} + \overbrace{d_e(v_i) + N_b(v_i)}^{\textit{vertices on polyhedron}} = E(v_i) + 2 \tag{1.7}$$

Note that this step assumes that the polyhedron is homeomorphic to a sphere (otherwise Euler's formula fails). In reality, this is not a severe assumption. (It would preclude a mesh consisting of two tetrahedra which touch at a single vertex.) On the polyhedron we also have that

$$2E(v_i) = 3\overbrace{(d_\phi(v_i) + F_b(v_i))}^{\text{polyhedral faces}}. \tag{1.8}$$

Combining (1.7) and (1.8) yields

$$d_e(v_i) = \frac{1}{2}d_\phi(v_i) + \frac{1}{2}F_b(v_i) - N_b(v_i) + 2. \tag{1.9}$$

Summing this equation over <u>all</u> vertices produces

$$\bar{d}_e n(v) = \frac{1}{2}\left(\bar{d}_\phi n(v) + \sum_i F_b(v_i)\right) - \sum_i N_b(v_i) \tag{1.10}$$

where $\bar{d}_e$ and $\bar{d}_\phi$ are the average vertex degrees with respect to edges and volumes. Since globally we have that

$$\bar{d}_e n(v) = 2n(e), \quad \bar{d}_\phi n(v) = 4n(\phi), \tag{1.11}$$

substitution of (1.11) into (1.10) reveals that

$$n(e) = n(\phi) + n(v) + \frac{1}{4}\sum F_b(v_i) - \frac{1}{2}\overbrace{\sum N_b(v_i)}^{n(v)_{bound}}. \qquad (1.12)$$

Finally, note that $\sum F_b(v_i) = 3n(f)_{bound}$. Inserting this relationship into (1.12) yields

$$n(e) = n(\phi) + n(v) + \frac{3}{4}n(f)_{bound} - \frac{1}{2}n(v)_{bound} \qquad (1.13)$$

Other equivalent formulas are easily obtained by combining this equation with the formula relating volume, faces, and boundary faces, i.e.

$$n(f) = 2n(\phi) + \frac{1}{2}n(f)_{bound} \qquad (1.14)$$

An exact formula, similar to (1.6), is obtained by combining (1.14) and (1.13)

$$n(e)+n(\phi)=n(f)+n(v)+\frac{1}{4}n(f)_{bound}-\frac{1}{2}n(v)_{bound} \qquad (1.15)$$

1.2 Duality

Given a planar graph G, we informally define a dual graph G_{Dual} to be any graph with the following three properties: each vertex of G_{Dual} is associated with a face of G; each edge of G is associated with an edge of G_{Dual}; if an edge separates two faces, f_i and f_j of G then the associated dual edge connects two vertices of G_{Dual} associated with f_i and f_j. This duality plays an important role in CFD algorithms.

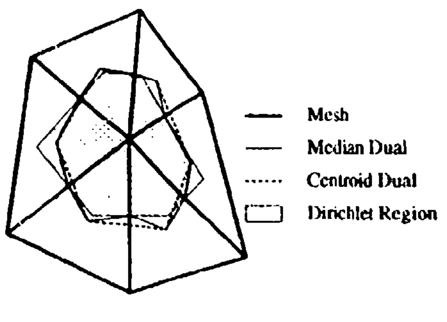


Figure 1.1 Several triangulation duals.

In figure 1.1, edges and faces about the central vertex are shown for duals formed from median segments, centroid segments, and by Dirichlet tessellation. (The Dirichlet tessellation of a set of points is a pattern of convex regions in the plane, each region being the portion of the plane closer to some given point P of the set of points than to any other point.) Two-dimensional finite-volume schemes for the Euler and Navier-Stokes equations are frequently developed which form control volumes from either faces (cells) of the mesh or faces of the mesh dual. Schemes which use the cells of the mesh as control volumes are often called "cell-centered" schemes. Other "vertex" schemes use mesh duals constructed from median segments, Dirichlet regions, or centroid segments. In all of these schemes, the primary computational effort is associated with the calculation of the flux of mass, momenta, and energy through an edge associated with the control volume. The one-to-one correspondence of edges of a mesh and mesh dual (ignoring boundaries) means that there is very little difference in computational effort in schemes based on mesh faces or duals. This observation is not true in three dimensions! Consider a three dimensional tetrahedral mesh. The duality for this nonplanar arrangement is between *edges* of the tetrahedral mesh and *faces* of the dual. In other words, for each edge of the mesh there is a one-to-one correspondence with a face of the dual (ignoring boundaries). Again, the main computational effort associated with finite-volume schemes for solving the Euler and Navier-Stokes equations is the calculation of the flux through each face of the control volume. If the control volumes are the tetrahedra themselves (cell-centered scheme), then a flux must be calculated for each tetrahedral face. This means that the work is proportional to the number of faces of the tetrahedral mesh. From eqn. (1.14), the number of faces of a tetrahedral mesh is related to the number of tetrahedra and boundary faces by

$$work_{c-c\ scheme} \propto n(f) = 2n(\phi) + \frac{1}{2}n(f)_{bound}.$$

If the control volumes of the finite-volume scheme are formed from a mesh dual (vertex scheme), then the number of flux calculations is proportional to the number of faces of the mesh dual which is roughly equal to the number of edges of the original tetrahedral mesh. From eqn. (1.13) we have that

$$work_{vert\ scheme} \propto n(e) = n(\phi) + n(v) + \frac{3}{4}n(f)_{bound} - \frac{1}{2}n(v)_{bound}$$

To better understand the work estimates, define β such that $n(\phi) = \beta n(v)$. Practically speaking, β usually ranges from 5-7 for tetrahedral meshes. Taking the ratio of the work estimates for the cell-centered and vertex scheme, ignoring boundary terms and assuming an identical constant of proportionality, we obtain

$$\frac{work_{c-c\ scheme}}{work_{vert\ scheme}} = \frac{2\beta n(v)}{(1+\beta)n(v)} = \frac{2\beta}{1+\beta} \quad (1.16)$$

The work for the cell-centered scheme approaches twice that of the vertex scheme. The reader should not automatically infer that the vertex scheme is preferred. The question of solution accuracy of the two approaches needs to be factored into the equation. The answer to the question of which is "better" is still a subject for debate.

1.3 Data Structures

The choice of data structures used in representing unstructured grids varies considerably depending on the type of algorithmic operations to be performed. In this section, a few of the most common data structures will be discussed. The mesh is assumed to have a numbering of vertices, edges, faces, etc. In most cases, the physical coordinates are simply listed by vertex number. The "standard" finite element (FE) data structure lists connectivity of each element. For example in figure 1.2(a), a list of the three vertices of each triangle would be given.

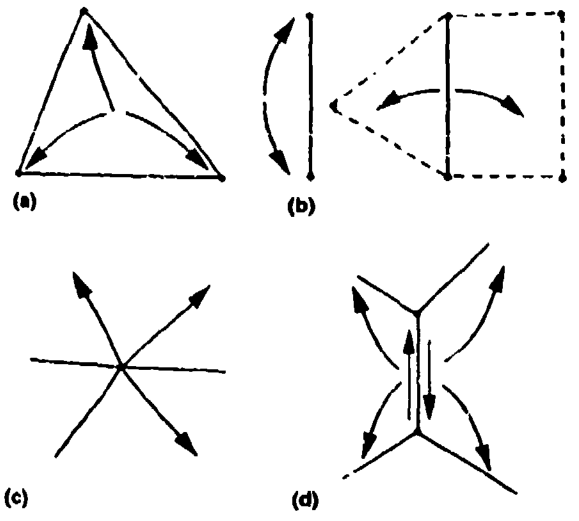


Figure 1.2 Data structures for planar graphs. (a) FE data structure, (b) Edge structure, (c) Out-degree structure, (d) Quad-edge structure.

The FE structure extends naturally to three dimensions. The FE structure is used extensive in finite element solvers for solids and fluids.

For planar meshes, another typical structure is the *edge structure* (figure 1.2(b)) which lists connectivity of vertices and adjacent faces by storing a quadruple for each edge consisting of the origin and destination of the each edge as well as the two faces (cells) that share that edge. This structure allows easy traversal through a mesh which can be useful in certain grid generation algorithms. (This traversal is not easily done using the FE structure.) The extension to three dimensions is facewise (vertices of a face are given as well as the two neighboring volumes) and requires distinction between different face types.

A third data structure provides connectivity via *vertex lists* as shown in figure 1.2(c). The brute force approach is to list all adjacent neighbors for each vertex (usually as a linked-list). Many sparse matrix solver packages specify nonzeros of a matrix using row or column storage schemes which list all nonzero entries of a given row or column. For discretizations involving only adjacent neighbors of a mesh, this would be identical to specifying a vertex list. An alternative to specifying all adjacent neighbors is to direct edges of the mesh. In this case only those edges which point outward from a vertex are listed. In the next section, it will be shown that an *out-degree list* can be constructed for planar meshes by directing a graph such that no vertex has more than three outgoing edges. This is asymptotically optimal. The extension of the out-degree structure to three dimensions is not straightforward and algorithms for obtaining optimal edge direction for nonplanar graphs are still under development.

The last structure considered here is the *quad-edge structure* proposed by Guibas and Stolfi [1], see figure 1.2(d). Each edge is stored as pair of directed edges. Each of the directed edges stores its origin and pointers to the next and previous directed edge of the region to its left. The quad-edge structure is extremely useful in grid generation where distinctions between topological and geometrical aspects are sometimes crucial. The structure has been extended to three dimensional arrangements by Dobkin and Laslo [2] and Brisson [3].

2.0 Some Basic Graph Operations Important in CFD

Implementation of unstructured grid methods on differing computer architectures has stimulated research in exploiting properties and characterizations of planar and nonplanar graphs. For example in Hammond and Barth [4], we exploited a recent theorem by Chrobak and Eppstein [5] concerning directed graphs with minimum out-degree. In this section, we review this result as well as presenting other basic graph operations that are particularly useful in CFD. Some of these algorithms are specialized to planar graphs (2-D meshes) while others are very general and apply in any number of space dimensions.

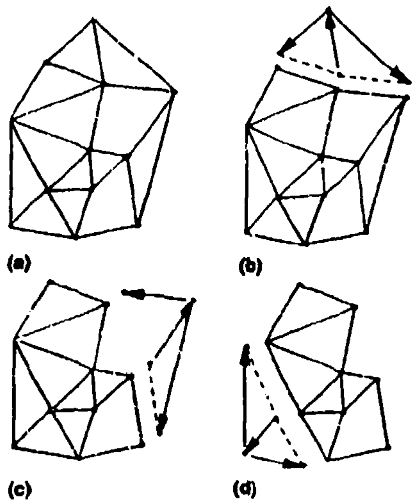
2.1 Planar Graphs with Minimum Out-Degree

Theorem: *Every planar graph has a 3-bounded orientation*, [5].

In other words, each edge of a planar graph can be assigned a direction such that the maximum number of edges pointing outward from any vertex is less than or equal to three. A constructive proof is given in ref.[5] consisting of the following steps. The first step is to find a *reduceable* boundary vertex. A reduceable boundary vertex is any vertex on the boundary with incident exterior (boundary) edges that connect to at most two other boundary vertices and any number of interior edges. Chrobak and Eppstein prove that reduceable vertices can always be found for arbitrary planar graphs. (In fact, two reduceable vertices can always be found!) Once a reduceable vertex is found then the two edges connecting to the other boundary vertices are directed *outward* and the remaining edges are always directed *inward*. These directed edges are then removed from the graph, see Figures 2.0(a-j). The process is then repeated on the remaining graph until no more edges remain. The algorithm shown pictorially in figure 2.0 is summarized in the following steps:

Algorithm: Orient a Graph with maximum out-degree ≤ 3.

Step 1. Find reduceable boundary vertex.
Step 2. Direct exterior edges outward and interior edges inward.
Step 3. Remove directed edges from graph.
Step 4. If undirected edges remain, go to step 1



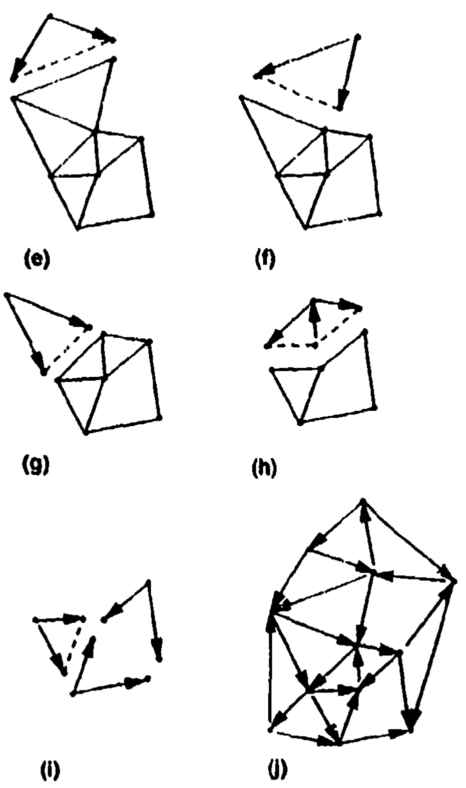


Figure 2.0 (a-i) Mesh orientation procedure with out-degree 3, (j) final oriented triangulation.

Linear time algorithms are given in [5]. In the paper by Hammond and Barth, we exploit the out-degree property to provide optimal load balancing on a massively parallel computer. Details are given in a later section.

2.2 Graph Ordering Techniques

The particular ordering of solution unknowns can have a marked influence on the amount of computational effort and memory storage required to solve large sparse linear systems and eigenvalue problems. In many algorithms, the band width and/or profile of the matrix determines the amount of computation and memory required. Most meshes obtained from grid generation codes have very poor natural orderings. Figures 2.1 and 2.2 show a typical mesh generated about a multi-component airfoil and the nonzero entries associated with the "Laplacian" of the graph. The Laplacian of a graph would represent the nonzero entries due to a discretization which involves only adjacent neighbors of the mesh. Figure 2.2 indicates that the band width of the natural ordering is almost equal to the dimension of the matrix! In parallel computation, the ordering algorithms can be used as means for *partitioning* a mesh among processors of the computer. This will be addressed in the next section.

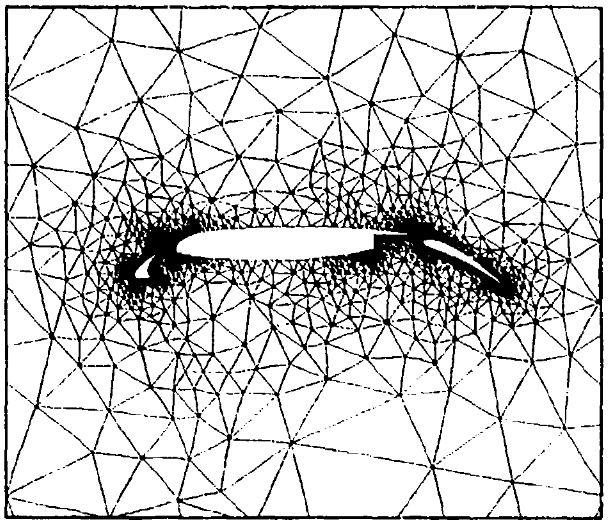


Figure 2.1 Typical Steiner triangulation about multi-component airfoil.

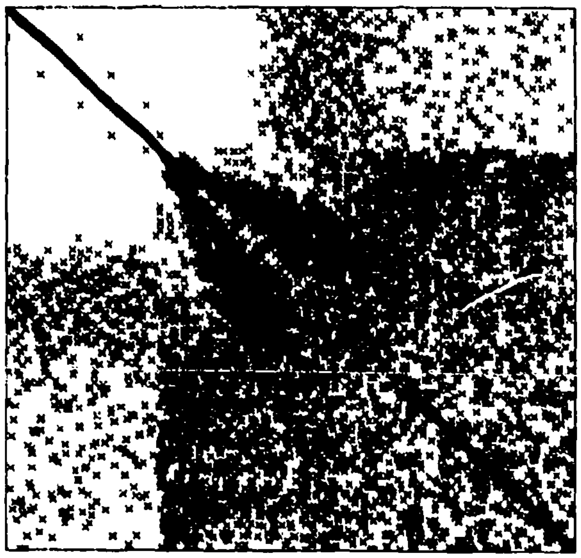


Figure 2.2 Nonzero entries of Laplacian matrix produced from natural ordering.

Several algorithms exist which construct new orderings by attempting to minimize the band width of a matrix or attempting to minimize the fill that occurs in the process matrix factorization. These algorithms usually rely on heuristics to obtain high efficiency, and do not usually obtain an optimum ordering. One example would be Rosen's algorithm [6] which iterates on the ordering to minimize the maximum band width.

Algorithm: Graph ordering, Rosen.

Step 1. Determine band width and the defining index pair (i, j) with $(i < j)$
Step 2. Does their exist an exchange which increases i or decreases j so that the band width is reduced? If so, exchange and go to step 1
Step 3. Does their exist an exchange which increases i or decreases j so that the band width remains the same? If so, exchange and go to step 1

This algorithm produces very good orderings but can be very expensive for large matrices. A popular method which is much less expensive for large matrices is the Cuthill-McKee [7] algorithm.

Algorithm: Graph ordering, Cuthill-McKee.

Step 1. Find vertex with lowest degree. This is the ***root*** vertex.
Step 2. Find all neighboring vertices connecting to the root by incident edges. Order them by increasing vertex degree. This forms level 1.
Step 3. Form level k by finding all neighboring vertices of level $k - 1$ which have not been previously ordered. Order these new vertices by increasing vertex degree.
Step 4. If vertices remain, go to step 3

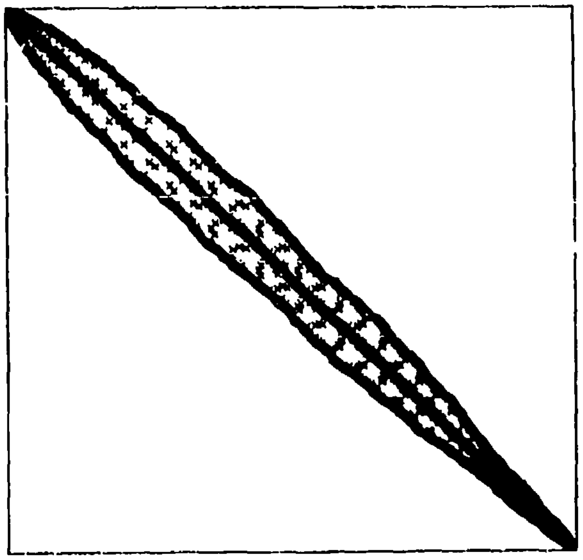


Figure 2.3 Nonzero entries of Laplacian matrix after Cuthill-McKee ordering.

The heuristics behind the Cuthill-McKee algorithm are very simple. In the graph of the mesh, neighboring vertices must have numberings which are near by, otherwise they will produce entries in the matrix with large band width. The idea of sorting elements among a given level is based on the heuristic that vertices with high degree should be given indices as large as possible so that they will be as close as possible to vertices of the *next* level generated. Figure 2.3 shows the dramatic improvement of the Cuthill-McKee ordering on the matrix shown in figure 2.2.

Studies of the Cuthill-McKee algorithm have shown that the profile of a matrix can be greatly reduced simply by reversing the ordering of the Cuthill-McKee algorithm, see George [8]. This amounts to a renumbering given by

$$k \rightarrow n - k + 1 \qquad (2.1)$$

where n is the size of the matrix. While this does not change the bandwidth of the matrix, it can dramatically reduce the fill that occurs in Cholesky or L-U matrix factorization when compared to the original Cuthill-McKee algorithm.

2.3 Graph Partitioning for Parallel Computing

An efficient partitioning of a mesh for distributed memory machines is one that ensures an even distribution of computational workload among the processors and minimizes the amount of time spent in interprocessor communications. The former requirement is termed *load balancing*. For if the load were not evenly distributed, some processors will have to sit idle at synchronization points waiting for other processors to catch up. The second requirement comes from the fact that communication between processors takes time and it is not always possible to hide this latency in data transfer. In our parallel implementation of a finite-volume flow solver on unstructured grids, data for the nodes that lie on the boundary between two processors is exchanged, hence requiring a bidirectional data-transfer. On many systems, a synchronous exchange of data can yield a higher performance than when done asynchronously. To exploit this fact, edges of the communication graph are colored such that no vertex has more than one edge of a certain color incident upon it. A communication graph is a graph in which the vertices are the processors and an edge connects two vertices if the two corresponding processors share an interprocessor boundary. The colors in the graph represent separate communication cycles. For instance, the mesh partitioned amongst four processors as shown in figure 2.4(a), would produce the communication graph shown in figure 2.4(b).

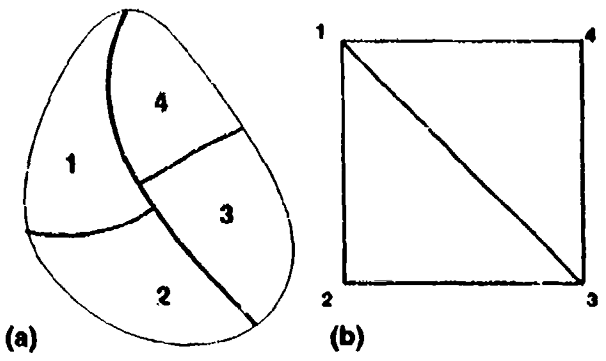


Figure 2.4 (a) Four partition mesh, (b) Communication graph.

The graph shown in figure 2.4(b) can be colored edgewise using three colors. For example, in the first communication cycle, processors $(1,4)$ could perform a synchronous data exchange as would processors $(2,3)$. In the second communication cycle, processors $(1,2)$ and $(3,4)$ would exchange and in the third cycle, processors $(1,3)$ would exchange while processors 2 and 4 sit idle. Vizing's theorem proves that any graph of maximum vertex degree Δ (number of edges incident upon a vertex) can be colored using n colors such that $\Delta \leq n \leq \Delta + 1$. Hence, any operation that calls for every processor to exchange data with its neighbors will require n communication cycles.

The actual cost of communication can often be accurately modeled by the linear relationship:

$$Cost = \alpha + \beta m \qquad (2.2)$$

where α is the time required to initiate a message, β is the rate of data-transfer between two processors and m is the message length. For n messages, the cost would be

$$Cost = \sum_{n} (\alpha + \beta m_n). \qquad (2.3)$$

This cost can be reduced in two ways. One way is to reduce Δ thereby reducing n. The alternative is to reduce the individual message lengths. The bounds on n are $2 \leq N \leq P-1$ for $P \geq 3$ where P is the total number of processors. Figure 2.5(a) shows the partitioning of a mesh which reduces Δ, and 2.5(b) shows a partitioning which minimizes the message lengths. For the mesh in figure 2.5(a), $\Delta = 2$ while in figure 2.5(b), $\Delta = 3$. However, the average message length for the partitions shown in figure 2.5(b) is about half as much as that in figure 2.5(a).

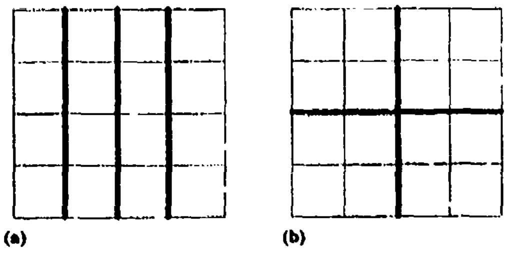


Figure 2.5 (a) Mesh partitioning with minimized Δ, (b) Mesh with minimizes message length.

In practice, it is difficult to partition an unstructured mesh while simultaneously minimizing the number and length of messages. In the following paragraphs, a few of the most popular partitioning algorithms which approximately accomplish this task will be discussed. All the algorithms discussed below: coordinate bisection, Cuthill-McKee, and spectral partitioning are evaluated in the paper by Venkatakrishnan, Simon, and Barth [9]. This paper evaluates the partitioning techniques within the confines of an explicit, unstructured finite-volume Euler solver. Spectral partitioning has been extensively studied by Simon [10]. The algorithms have also been implemented in three dimensions by A. Gandhi working in the CFD branch at NASA Ames Research Center.

Note that for the particular applications that we have in mind (a finite-volume scheme with solution unknowns at vertices of the mesh), it makes sense to partition the domain such that the separators correspond to edges of the mesh. Also note that the partitioning algorithms all can be implemented recursively. The mesh is first divided into two sub-meshes of nearly equal size. Each of these sub-meshes is subdivided into two more sub-meshes and the process in repeated until the desired number of partitions P is obtained (P is a integer power of 2). Since we desire the separator of the partitions to coincide with edges of the mesh, the division of a sub-mesh into two pieces can be viewed as a 2-coloring of *faces* of the sub-mesh. For the Cuthill-McKee and spectral partitioning techniques, this amounts to supplying these algorithms with the *dual* of the graph for purposes of the 2-coloring. The balancing of each partition is usually done cellwise; although an edgewise balancing is more appropriate in the present applications. Due to the recursive nature of partitioning, the algorithms outlined below represent only a single step of the process.

Coordinate Bisection

In the coordinate bisection algorithm, face centroids are sorted either horizontally or vertically depending of the current level of the recursion. A separator is chosen which balances the number of faces. Faces are colored depending on which side of the separator they are located. The actual edges of the mesh corresponding to the separator are characterized as those edges which have adjacent faces of different color, see figure 2.6. This partitioning is very efficient to create but gives sub-optimal performance on parallel computations owing to the long message lengths than can routinely occur.

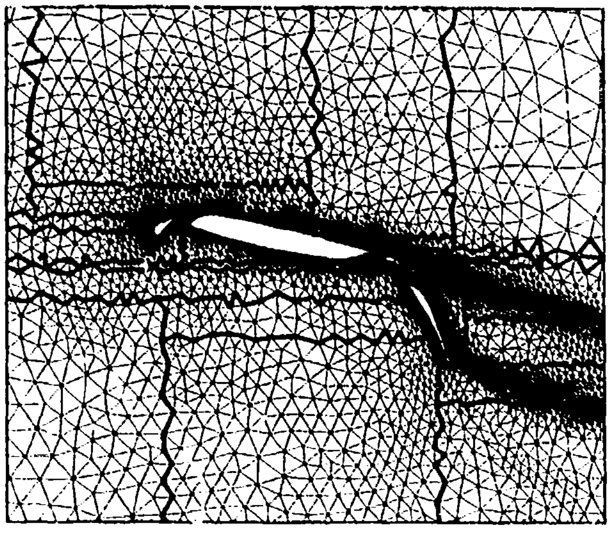


Figure 2.6 Coordinate bisection (16 partitions).

Cuthill-McKee

The Cuthill-McKee (CM) algorithm described earlier can also be used for recursive mesh partitioning. In this case, the Cuthill-McKee ordering is performed on the *dual* of the mesh graph. A separator is chosen either at the median of the ordering (which would balance the coloring of faces of the original mesh) or the separator is chosen at the level set boundary *closest* to the median as possible. This latter technique has the desired effect of reducing the number of disconnected sub-graphs that occur during the course of the partitioning. Figure 2.7 shows a Cuthill-McKee partitioning for the multi-component airfoil mesh. The Cuthill-McKee ordering tends to produce long boundaries because of the way that the ordering is propagated through a mesh. The maximum degree of the communication graph also tends to be higher using the Cuthill-McKee algorithm. The results shown in ref. [9] for multi-component airfoil grids indicate a performance on parallel computations which is slightly worse than the coordinate bisection technique.

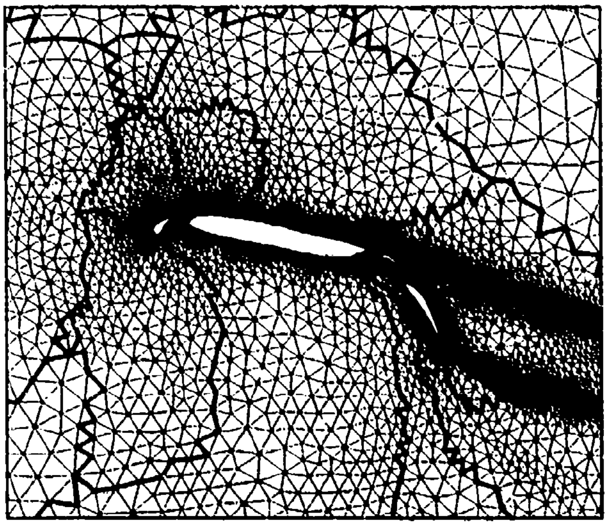


Figure 2.7 Cuthill-McKee partitioning of mesh.

Spectral Partitioning

The last partitioning considered is the spectral partitioning which exploits properties of the Laplacian $\mathcal{L}$ of a graph (defined below). The algorithm consists of the following steps:

Algorithm: Spectral Partitioning.

Step 1. Calculate the matrix $\mathcal{L}$ associated with the Laplacian of the graph (dual graph in the present case).
Step 2. Calculate the eigenvalues and eigenvectors of $\mathcal{L}$.
Step 3. Order the eigenvalues by magnitude, $\lambda_1 \leq \lambda_2 \leq \lambda_3 ... \lambda_N$.
Step 4. Determine the smallest nonzero eigenvalue, λ_f and its associated eigenvector $\mathbf{x}_f$ (the Fiedler vector).
Step 5. Sort elements of the Fiedler vector.
Step 6. Choose a divisor at the median of the sorted list and 2-color vertices of the graph (or dual) which correspond to elements of the Fielder vector less than or greater than the median value.

The spectral partitioning of the multi-component airfoil is shown in figure 2.8. In reference [9], we found that parallel computations performed slightly better on the spectral partitioning than on the coordinate bisection or Cuthill-McKee. The cost of the spectral partitioning is very high (even using a Lanczos algorithm to compute the eigenvalue problem). It has yet to be determined if the spectral partitioning will have practical merit.

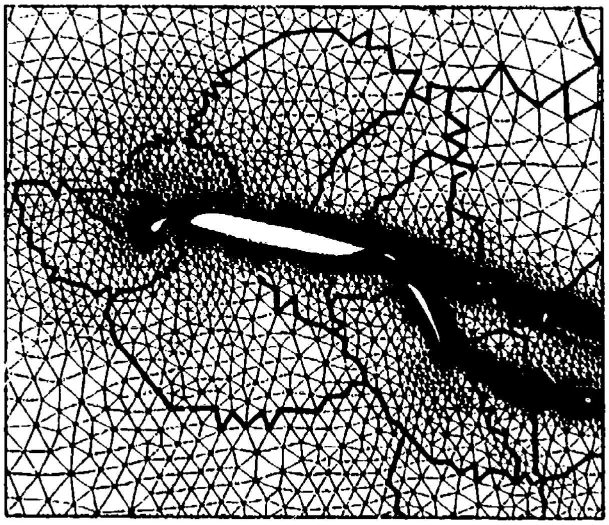


Figure 2.8 Spectral partitioning of multi- component airfoil.

The spectral partitioning exploits a peculiar property of the "second" eigenvalue of the Laplacian matrix associated with a graph. The Laplacian matrix of a graph is simply

$$\mathcal{L} = -\mathcal{D} + \mathcal{A}. \tag{2.4}$$

where $\mathcal{A}$ is the standard adjacency matrix

$$\mathcal{A}_{ij} = \begin{cases} 1 & e(v_i, v_j) \in G \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

and $\mathcal{D}$ is a diagonal matrix with entries equal to the degree of each vertex, $\mathcal{D}_i = d(v_i)$. From this definition, it should be clear that rows of $\mathcal{L}$ each sum to zero. Define an N-vector, $\mathbf{s} = [1, 1, 1, ...]^T$. By construction we have that

$$\mathcal{L}\mathbf{s} = 0. \tag{2.6}$$

This means that at least one eigenvalue is zero with $\mathbf{s}$ as an eigenvector.
The objective of the spectral partitioning is to divide the mesh into two partitions of equal size such that the number of edges cut by the partition boundary is approximately minimized.

Technically speaking, the smallest nonzero eigenvalue need not be the second. Graphs with disconnected regions will have more that one zero eigenvalue depending on the number of disconnected regions. For purposes of discussion, we assume that disconnected regions are not present, i.e. that λ_2 is the relevant eigenmode.

Elements of the proof:

Define a partitioning vector which 2-colors the vertices

$$\mathbf{p} = [+1, -1, -1, +1, +1, \ldots, +1, -1]^T \qquad (2.7)$$

depending on the sign of elements of $\mathbf{p}$ and the one-to-one correspondence with vertices of the graph, see for example figure 2.9. Balancing the number of vertices of each color amounts to the requirement that

$$\mathbf{s} \perp \mathbf{p} \qquad (2.8)$$

where we have assumed an even number of vertices.

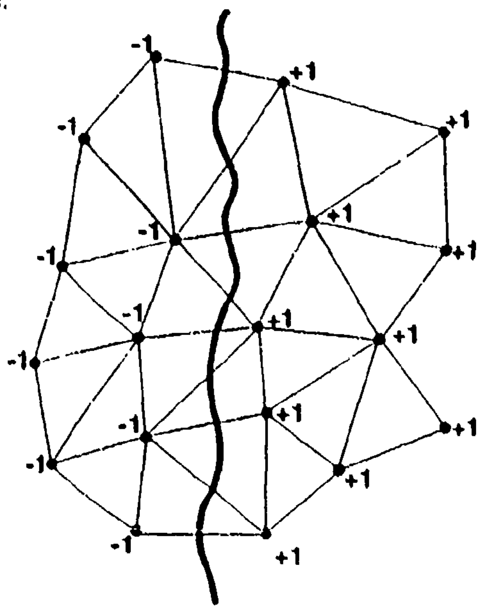


Figure 2.9 Arbitrary graph with 2-coloring showing separator and cut edges.

The key observation is that the number of cut edges, E_c, is precisely related to the L_1 norm of the Laplacian matrix multiplying the partitioning vector, i.e.

$$4E_c = \|\mathcal{L}\mathbf{p}\|_1 \qquad (2.9)$$

which can be easily verified. The goal is to minimize cut edges. That is to find $\mathbf{p}$ which minimizes $\|\mathcal{L}\mathbf{p}\|_1$ subject to the constraints that $\|\mathbf{p}\|_1 = N$ and $\mathbf{s} \perp \mathbf{p}$. Since $\mathcal{L}$ is a real symmetric (positive semi-definite) matrix, it has a complete set of real eigenvectors which can be orthogonalized with each other. The next step of the proof would be to extend the domain of $\mathbf{p}$ to include real numbers (this introduces an inequality) and expand $\mathbf{p}$ in terms of the orthogonal eigenvectors.

$$\mathbf{p} = \sum_{i=1}^{n} c_i \mathbf{x}_i \qquad (2.10)$$

By virtue of (2.6) we have that $\mathbf{x}_1 = \mathbf{s}$. It remains to be shown that $\|\mathcal{L}\mathbf{p}\|_1$ is minimized when $\mathbf{p} = \mathbf{p}' = n\mathbf{x}_2/\|\mathbf{x}_2\|_1$,i.e. when the Fiedler vector is used. Inserting this expression for $\mathbf{p}$ we have that

$$\|\mathcal{L}\mathbf{p}'\|_1 = n\lambda_2 \qquad (2.11)$$

It is a simple matter to show that adding any other eigenvector component to $\mathbf{p}'$ while insisting that $\|\mathbf{p}\|_1 = N$ can only increase the L_1 norm. This would complete the proof. Figure 2.10 plots contours (level sets) of the Fiedler vector for the multi-component airfoil problem.

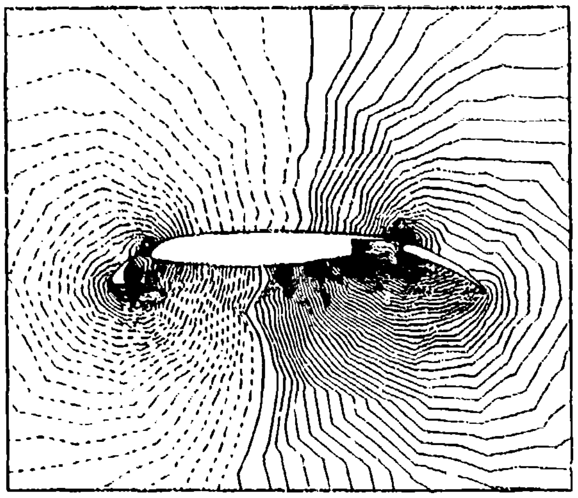


Figure 2.10. Contours of Fiedler Vector for Spectral Partitioning. Dashed lines are less than the median value.

3.0 Triangulation Methods

Although many algorithms exist for triangulating sites (points) in an arbitrary number of space dimensions, only a few have been used on practical problems. In particular, Delaunay triangulation has proven to be a very useful triangulation technique. This section will present some of the basic concepts surrounding Delaunay and related triangulations as well as discussing some of the most popular algorithms for constructing these triangulations. The discussion of the advancing front method of grid generation will be deferred to Professors Morgan and Löhner.

3.1 Voronoi Diagram and Delaunay Triangulation

Recall the definition of the Dirichlet tessellation in a plane. The Dirichlet tessellation of a point set is the pattern of convex regions, each being closer to some point P in the point set than to any other point in the set. These Dirichlet regions are also called Voronoi regions. The edges of Voronoi polygons comprise the Voronoi diagram, see figure 3.1. The idea extends naturally to higher dimensions.

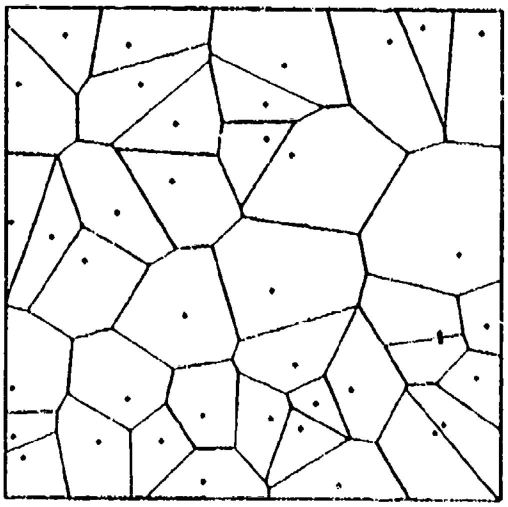


Figure 3.1 Voronoi diagram of 40 random sites.

Voronoi diagrams have a rich mathematical theory. *The Voronoi diagram is believed to be one of the most fundamental constructs defined by discrete data.* Voronoi diagrams have been independently discovered in a wide variety of disciplines. Computational geometricians have a keen interest in Voronoi diagrams. It is well known that Voronoi diagrams are related to convex hulls via stereographic projection. Point location in a Voronoi diagram can be performed in $O(\log(n))$ time with $O(n)$ storage for n regions. This is useful in solving post-office or related problems in optimal time. Another example of the Voronoi diagram which occurs in the natural sciences can be visualized by placing crystal "seeds" at random sites in 3-space. Let the crystals grow at the same rate in all directions. When two crystals collide simply stop their growth. The crystal formed for each site would represent that volume of space which is closer to that site than to any other site. This would effectively construct a Voronoi diagram. We now consider the role of Voronoi diagrams in Delaunay triangulation.

Definition: The Delaunay triangulation of a point set is defined as the dual of the Voronoi diagram of the set.

The Delaunay triangulation in two space dimensions is formed by connecting two points if and only if their Voronoi regions have a common border segment. If no four or more points are cocircular, then we have that *vertices of the Voronoi are circumcenters of the triangles.* This is true because vertices of the Voronoi represent locations that are equidistant to three (or more) sites. Also note that from the definition of duality, edges of the Voronoi are in one-to-one correspondence to edges of the Delaunay triangulation (ignoring boundaries). Because edges of the Voronoi diagram are the locus of points equidistant to two sites, each edge of the Voronoi diagram is perpendicular to the corresponding edge of the Delaunay triangulation. This duality extends to three dimensions in a straightforward way. The Delaunay triangulation possesses several alternate characterizations and many properties of importance. Unfortunately, not all of the two dimensional characterizations have three-dimensional extensions. To avoid confusion, properties and algorithms for construction of two dimensional Delaunay triangulations will be considered first. The remainder of this section will then discuss the three-dimensional Delaunay triangulation.

3.2 Properties of a 2-D Delaunay Triangulation

(1) *Uniqueness.* The Delaunay triangulation is unique. This assumes that no four sites are cocircular. The uniqueness follows from the uniqueness of the Dirichlet tessellation.

(2) *The circumcircle criteria.* A triangulation of $N \geq 2$ sites is Delaunay if and only if the circumcircle of every interior triangle is point-free. For if this was not true, the Voronoi regions associated with the dual would not be convex and the Dirichlet tessellation would be invalid. Related to the circumcircle criteria is the *incircle* test for four points as shown in figures 3.2-3.3.

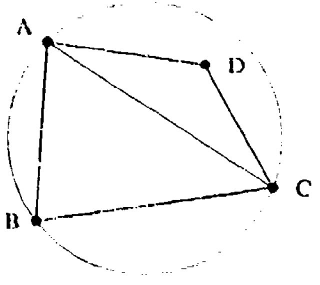


Figure 3.2 Incircle test for $\triangle ABC$ and D (true).

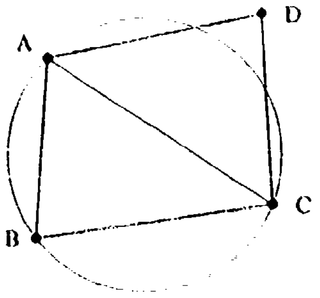


Figure 3.3 Incircle test for $\triangle ABC$ and D (false).

This test is true if point D lies interior to the circumcircle of $\triangle ABC$ which is equivalent to testing whether $\angle ABC + \angle CDA$ is less than or greater than $\angle BCD + \angle BAD$. More precisely we have that

$$\angle ABC + \angle CDA = \begin{cases} < 180^\circ & \text{incircle false} \\ 180^\circ & \text{A,B,C,D cocircular} \\ > 180^\circ & \text{incircle true} \end{cases} \qquad (3.1)$$

Since interior angles of the quadrilateral sum to 360°, if the circumcircle of $\triangle ABC$ contains D then swapping the diagonal edge from position $A-C$ into $B-D$ guarantees that the new triangle pair satisfies the circumcircle criteria. Furthermore, this process of diagonal swapping is local, i.e. it does not disrupt the Delaunayhood of any triangles adjacent to the quadrilateral.

(3)*Edge circle property.* A triangulation of sites is Delaunay if and only if there exists *some* circle passing through the endpoints of each and every edge which is point-free. This characterization is very useful because it also provides a mechanism for defining a *constrained* Delaunay triangulation where certain edges are prescribed *apriori*. A triangulation of sites is a constrained Delaunay triangulation if for each and every edge of the mesh there exists some circle passing through its endpoints containing no other site in the triangulation which is *visible* to the edge. In figure 3.4, site d is not visible to the segment a-c because of the constrained edge a-b.

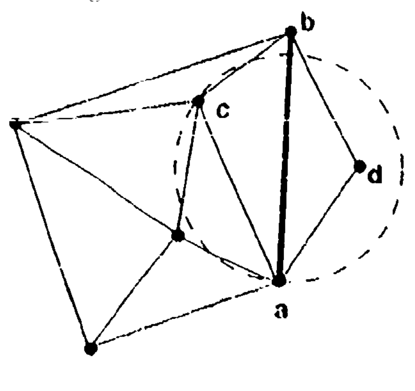


Figure 3.4 Constrained Delaunay triangulation. Site d is not visible to a-c due to constrained segment a-b.

(4)*Equiangularity property.* Delaunay triangulation maximizes the minimum angle of the triangulation. For this reason Delaunay triangulation often called the MaxMin triangulation. This property is also locally true for all adjacent triangle pairs which form a convex quadrilateral. This is the basis for the local edge swapping algorithm of Lawson [11] described below.

(5)*Minimum Containment Circle.* A recent result by Rajan [12] shows that the Delaunay triangulation minimizes the maximum containment circle over the entire triangulation. The containment circle is defined as the smallest circle enclosing the three vertices of a triangle. This is identical to the circumcircle for acute triangles and a circle with diameter equal to the longest side of the triangle for obtuse triangles (see figure 3.5).

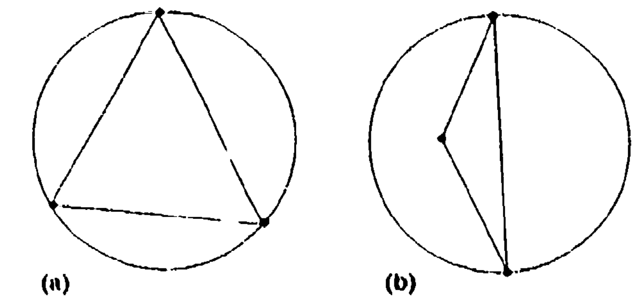


Figure 3.5 Containment circles for acute and obtuse triangles.

This property extends to n dimensions. Unfortunately, the result does not hold lexicographically.

(6)*Nearest neighbor property.* An edge formed by joining a vertex to its nearest neighbor is an edge of the Delaunay triangulation. This property makes Delaunay triangulation a powerful tool in solving the closest proximity problem. Note that the nearest neighbor edges do not describe all edges of the Delaunay triangulation.

(7)*Minimal roughness.* The Delaunay triangulation is a minimal roughness triangulation for arbitrary sets of scattered data, Rippa [13]. Given arbitrary data f_i at all vertices of the mesh and a triangulation of these points, a unique piecewise linear interpolating surface can be constructed. The Delaunay triangulation has the property that of all triangulations it minimizes the roughness of this surface as measured by the following Sobolev semi-norm:

$$\int_T \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 \right] dx\, dy \qquad (3.2)$$

This is a interesting result as it does not depend on the actual form of the data. This also indicates that Delaunay triangulation approximates well those functions which minimize this Sobolev norm. One example would be the harmonic functions satisfying Laplace's equation with suitable boundary conditions which minimize exactly this norm. In a later section, we will prove that a Delaunay triangulation guarantees a maximum principle for the discrete Laplacian approximation (with linear elements).

3.3 Algorithms for 2-D Delaunay Triangulation

We now consider several techniques for Delaunay triangulation in two dimensions. These methods were chosen because they perform optimally in rather different situations. The discussion of the 2-D algorithms is organized as follows:

(a) Incremental Insertion Algorithms
- (i) Bowyer algorithm
- (ii) Watson algorithm
- (iii) Green and Sibson algorithm

(b) Divide and Conquer Algorithm

(c) Tanemura/Merriam Algorithm

(d) Global Edge Swapping (Lawson)

It should be pointed out that there appears to be some confusion in the CFD literature concerning the Bowyer[14] and Watson[15] algorithms. What is sometimes described as Bowyer's algorithm is actually Watson's algorithm. This is surprising since both the Bowyer and Watson algorithms appeared as back-to-back articles in the same journal! The fundamental difference (as we will see) is that the Bowyer algorithm is implemented in the *Voronoi* plane and the Watson algorithm is implemented in the *triangulation* plane.

3.3a Incremental Insertion Algorithms

For simplicity, assume that the site to be added lies within a bounding polygon of the existing triangulation. If we desire a triangulation from a new set of sites, three initial phantom points can always be added which define a triangle large enough to enclose all points to be inserted. In addition, interior boundaries are usually temporarily ignored for purposes of the Delaunay triangulation. After completing the triangulation, spurious edges are then deleted as a postprocessing step. Incremental insertion algorithms begin by inserting a new site into an existing Delaunay triangulation. This introduces the task of point location in the triangulation. Some incremental algorithms require knowing which triangle the new site falls within. Other algorithms require knowing *any* triangle whose circumcircle contains the new site. In either case, two extremes arise in this reguard. Typical mesh adaptation and refinement algorithms determine the particular cell for site insertion as part of the mesh adaptation algorithm, thereby reducing the burden of point location. In the other extreme, initial triangulations of randomly distributed sites usually require advanced searching techniques for point location to achieve asymptotically optimal complexity $O(N \log N)$. Search algorithms based on quad-tree and split-tree data structures work extremely well in this case. Alternatively, search techniques based on "walking" algorithms are frequently used because of their simplicity. These methods work extremely well when successively added points are close together. The basic idea is start at the location in the mesh of the previously inserted point and move one edge (or cell) at a time in the general direction of the newly added point. In the worst case, each point insertion requires $O(N)$ walks. This would result in a worst case overall complexity $O(N^2)$. For randomly distributed points, the average point insertion requires $O(N^{\frac{1}{2}})$ walks which gives an overall complexity $O(N^{\frac{3}{2}})$. For many applications where successive points tend to be close together, the number of walks is roughly constant and these simple algorithms can be very competitive. Using any of these techniques, we can proceed with the insertion algorithms.

Bowyer's algorithm

The basic idea in Bowyer's algorithm is to insert a new site into an existing Voronoi diagram (for example site Q in figure 3.6), determine its territory (dashed line in figure 3.6), delete any edges completely contained in the territory, then add new edges and reconfigure existing edges in the diagram. The following is Bowyer's algorithm essentially as presented by Bowyer (see reference [14] for complete details):

Algorithm: Incremental Delaunay triangulation, Bowyer [14].

Step 1. Insert new point (site) Q into the Voronoi diagram.

Step 2. Find any existing vertex in the Voronoi diagram closer to the new point than to its forming points. This vertex will be deleted in the new Voronoi diagram.

Step 3. Perform tree search to find remaining set of deletable vertices $\mathcal{V}$ that are closer to the new point than to their forming points. (In figure 3.6 this would be the set $\{v_3, v_4, v_5\}$)

Step 4. Find the set $\mathcal{P}$ of forming points corresponding to the deletable vertices. In figure 3.6, this would be the set $\{p_2, p_3, p_4, p_5, p_7\}$.

Step 5. Delete edges of the Voronoi which can be described by pairs of vertices in the set $\mathcal{V}$ if both forming points of the edges to be deleted are contained in $\mathcal{P}$

Step 6. Calculate the new vertices of the Voronoi, compute their forming points and neighboring vertices, and update the Voronoi data structure.

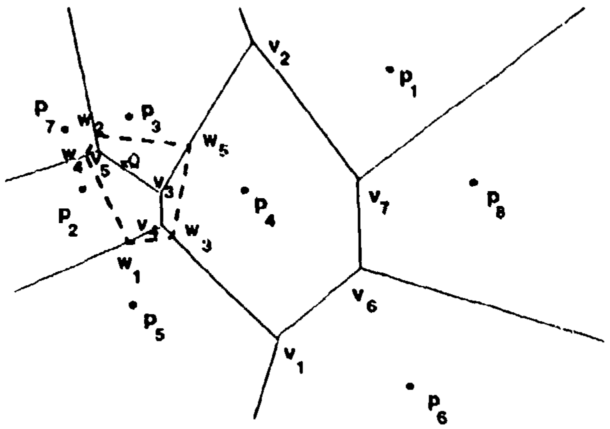


Figure 3.6 Voronoi diagram modified by Bowyer.

Implementational details and suggested data structures are given in the paper by Bowyer.

Watson's algorithm

Implementation of the Watson [15] algorithm is relatively straightforward. The first step is to insert a new site into an existing Delaunay triangulation and to find *any* triangle (the root) such that the new site lies interior to that triangles circumcircle. Starting at the root, a tree search is performed to find all triangles with circumcircle containing the new site. This is accomplished by recursively checking triangle neighbors. (The resulting set of deletable triangles violating the circumcircle criteria is independent of the starting root.) Removal of the deletable triangles exposes a polygonal cavity surrounding site Q with all the vertices of the polygon visible to site Q. The interior of the cavity is then retriangulated by connecting the vertices of the polygon to site Q, see figure 3.7(b). This completes the algorithm. A thorough account of Watson's algorithm is given by Baker [16] where he considers issues associated with constrained triangulations.

Algorithm: Incremental Delaunay triangulation, Watson [15].

Step 1. Insert new site Q into existing Delaunay triangulation.
Step 2. Find any triangle with circumcircle containing site Q.
Step 3. Perform tree search to find remaining set of deletable triangles with circumcircle containing site Q.
Step 4. Construct list of edges associated with deletable triangles. Delete all edges from the list that appear more that once.

Step 5. Connect remaining edges to site Q and update Delaunay data structure.

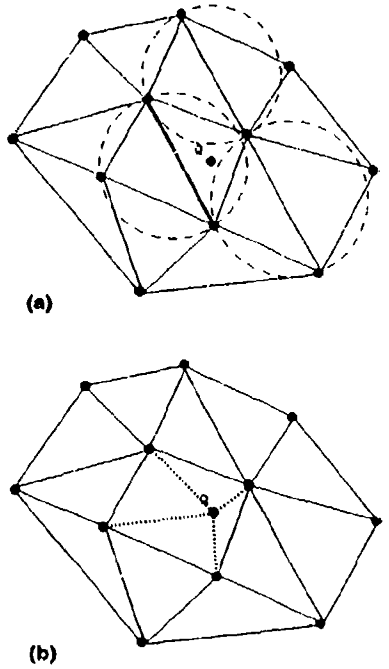
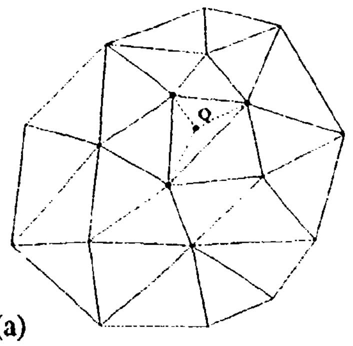


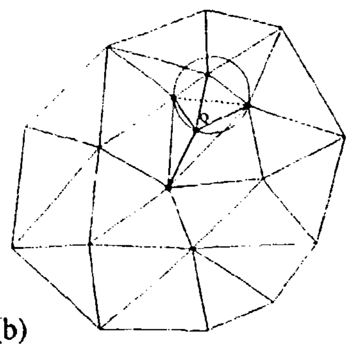
Figure 3.7 (a) Delaunay triangulation with site Q added. (b) Triangulation after deletion of invalid edges and reconnection.

Green and Sibson algorithm

The algorithm due to Green and Sibson [17] is very similar to the Watson algorithm. The primary difference is the use of local edge swapping to reconfigure the triangulation. The first step is location, i.e. find the triangle containing point Q. Once this is done, three edges are then created connecting Q to the vertices of this triangle as shown in figure 3.8(a). If the point falls on an edge, then the edge is deleted and four edges are created connecting to vertices of the newly created quadrilateral. Using the circumcircle criteria it can be shown that the newly created edges (3 or 4) are automatically Delaunay. Unfortunately, some of the original edges are now incorrect. We need to somehow find all "suspect" edges which could possibly fail the circle test. Given that this can be done (described below), each suspect edge is viewed as a diagonal of the quadrilateral formed from the two adjacent triangles. The circumcircle test is applied to either one of the two adjacent triangles of the quadrilateral. If the fourth point of the quadrilateral is interior to this circumcircle, the suspect edge is then swapped as shown in figure 3.8(b), two more edges then become suspect. At any given time we can immediately identify all suspect edges. To do this, first consider the subset of all triangles which share Q as a vertex. One can guarantee at all times that all initial edges incident to Q are Delaunay and any edge made incident to Q by swapping must be Delaunay. Therefore, we need only consider the remaining edges of this subset which form a polygon about Q as suspect and subject to the incircle test. The process terminates when all suspect edges pass the circumcircle test.



(a)



(b)

Figure 3.8 (a) Inserting of new vertex, (b) Swapping of suspect edge.

The algorithm can be summarized as follows:

Algorithm: Incremental Delaunay Triangulation, Green and Sibson [17]
Step 1. Locate existing cell enclosing point Q.
Step 2. Insert site and connect to 3 or 4 surrounding vertices.
Step 3. Identify suspect edges.
Step 4. Perform edge swapping of all suspect edges failing the incircle test.
Step 5. Identify new suspect edges.
Step 6. If new suspect edges have been created, go to step 3.

The Green and Sibson algorithm can be implemented using standard recursive programming techniques. The heart of the algorithm is the recursive procedure which would take the following form for the configuration shown in figure 3.9:

```
procedure swap[ $v_q$, $v_1$, $v_2$, $v_3$, edges]
if(incircle[$v_q$,$v_1$,$v_2$,$v_3$] = TRUE)then
    call reconfig_edges[$v_q$, $v_1$, $v_2$, $v_3$, edges]
    call swap[$v_q$, $v_1$, $v_4$, $v_2$, edges]
    call swap[$v_q$, $v_2$, $v_5$, $v_3$, edges]
endif
endprocedure
```

This example illustrates an important point. The nature of Delaunay triangulation guarantees that any edges swapped incident to Q will be final edges of the Delaunay triangulation. This means that we need only consider *forward propagation* in the recursive procedure. In a later section, we will consider incremental insertion and edge swapping for generating non-Delaunay triangulations based on other swapping criteria. This algorithm can also be programmed recursively but requires *backward propagation* in the recursive implementation. For the Delaunay triangulation algorithm, the insertion algorithm would simplify to the following three steps:

Recursive Algorithm: Incremental Delaunay Triangulation, Green and Sibson
Step 1. Locate existing cell enclosing point Q.
Step 2. Insert site and connect to surrounding vertices.
Step 3. Perform recursive edge swapping on newly formed cells (3 or 4).

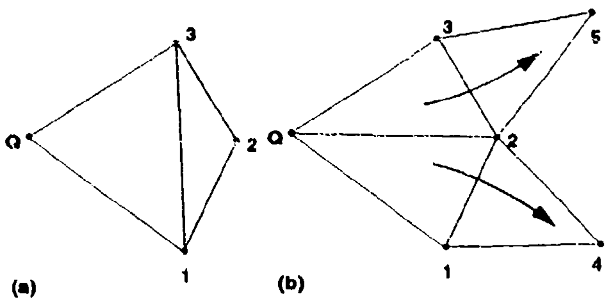


Figure 3.9 Edge swapping with forward propagation.

3.3b Divide-and-Conquer Algorithm

In this algorithm, the sites are assumed to be *prespecified*. The idea is to partition the cloud of points T (sorted along a convenient axis) into left (L) and right (R) half planes. Each half plane is then recursively Delaunay triangulated. The two halves must then be merged together to form a single Delaunay triangulation. Note that we assume that the points have been sorted along the x-axis for purposes of the following discussion (this can be done with $O(N \log N)$ complexity).

Algorithm: Delaunay Triangulation via Divide-and-Conquer
Step 1. Partition T into two subsets T_L and T_R of nearly equal size.
Step 2. Delaunay triangulate T_L and T_R recursively.
Step 3. Merge T_L and T_R into a single Delaunay triangulation.

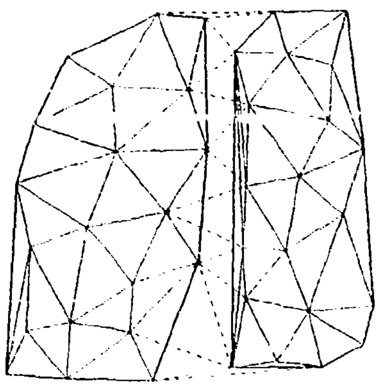


Figure 3.10 Triangulated subdivisions.

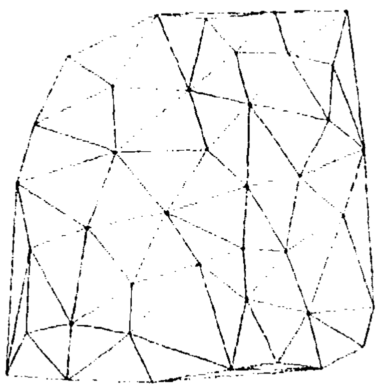


Figure 3.11 Triangulation after merge.

The only difficult step in the divide-and-conquer algorithm is the merging of the left and right triangulations. The process is simplified by noting two properties of the merge:

(1) *Only cross edges (L-R or R-L) are created in the merging process.* Since vertices are neither added or deleted in the merge process, the need for a new R-R or L-L edge indicates that the original right or left triangulation was defective. (Note that the merging process will require the deletion of edges L-L and/or R-R.)

(2) *Vertices with minimum (maximum) y value in the left and right triangulations always connect as cross edges.* This is obvious given that the Delaunay triangulation produces the convex :. il of the point cloud.

Given these properties we now outline the "rising bubble" [1] merge algorithm. This algorithm produces cross edges in ascending y-order. The algorithm begins by forming a cross edge by connecting vertices of the left and right triangulations with minimum y value (property 2). This forms the initial cross edge for the rising bubble algorithm. More generally consider the situation in which we have a cross edge between A and B and all edges incident to the points A and B with endpoints above the half plane formed by a line passing through $A - B$, see figure 3.12.

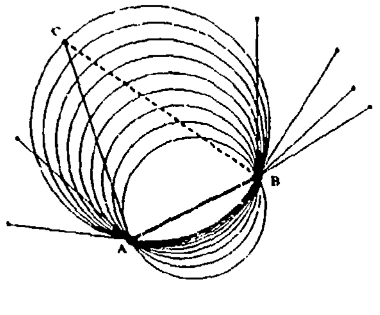


Figure 3.12 Circle of increasing radius in rising bubble algorithm.

This figure depicts a continuously transformed circle of increasing radius passing through the points A and B. Eventually the circle increases in size and encounters a point C from the left or right triangulation (in this case, point C is in the left triangulation). A new cross edge (dashed line in figure 3.12) is then formed by connecting this point to a vertex of $A - B$ in the other half triangulation. Given the new cross edge, the process can then be repeated and terminates when the top of the two meshes is reached. The deletion of $L - L$ or $R - R$ edges can take place during or after the addition of the cross edges. Properly implemented, the merge can be carried out in linear time, $O(N)$. Denoting $T(N)$ as the total running time, step 2 is completed in approximately $2T(N/2)$. Thus the total running time is described by the recurrence $T(N) = 2T(N/2) + O(N) = O(N \log N)$.

<u>3.3c Tanemura/Merriam Algorithm</u>

Another algorithm for performing Delauna triangulation is the advancing front method de veloped by Tanemura, Ogawa, and Ogita [18] an later rediscovered by Merriam [19]. Here the ide is to start with a known boundary edge and forr a new triangle by joining both endpoints to one of the interior points. This may generate up to two additional edges, providing they aren't already part of another triangle. After all the boundary edges have been incorporated into triangles, the new edges will appear to be a (somewhat ragged) boundary. This moving boundary is often called an advancing front. The process continues until the front vanishes. The problem here is to make the triangulation Delaunay. This can be done by taking advantage of the *incircle* property; the circumcircle of a Delaunay triangle contains no other points. This allows the appropriate point to be selected iteratively as shown in Fig. 3.13.

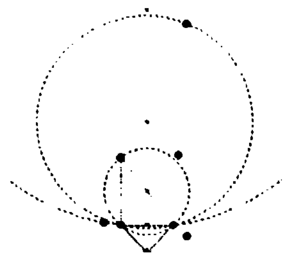


Figure 3.13 A straightforward iteration procedure selects the node which generates the smallest circumcircle for a given edge. The absence of nodes inside the circumcircle establishes convergence.

The iteration begins by selecting any node which is on the desired side of the given edge. If there are no such nodes, the given edge is part of a convex hull. Next, the circumcircle is constructed which passes through the edge endpoints and the selected node. Finally, check for nodes inside this circle. If there are any, replace the selected node with the node closest to the circumcenter and repeat the process. When the circumcircle is empty of nodes, connect the edge endpoints to the selected node.

3.3d Delaunay Triangulation Via Edge Swapping

This algorithm due to Lawson [11] assumes that a triangulation exists (not Delaunay) then makes it Delaunay through application of edge swapping such that the equiangularity of the triangulation increases. The equiangularity of a triangulation, $A(T)$, is defined as the ordering of angles $A(T) = [\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{3n(c)_3}]$ such that $\alpha_i \leq \alpha_j$ if $i < j$. We write $A(T^*) < A(T)$ if $\alpha_j^* \leq \alpha_j$ and $\alpha_i^* = \alpha_i$ for $1 \leq i < j$. A triangulation T is globally equiangular if $A(T^*) \leq A(T)$ for all triangulations T^* of the point set. Lawson's algorithm examines all interior edges of the mesh. Each of these edges represents the diagonal of the quadrilateral formed by the union of the two adjacent triangles. In general one must first check if the quadrilateral is convex so that a potential diagonal swapping can place without edge crossing. If the quadrilateral is convex then the diagonal position is chosen which optimizes a local criterion (in this case the local equiangularity). This amounts to maximizing the minimum angle of the two adjacent triangles. Lawson's algorithm continues until the mesh is locally optimized and locally equiangular everywhere. It is easily shown that the condition of local equiangularity is equivalent to satisfaction of the incircle test described earlier. Therefore a mesh which is locally equiangular everywhere is a Delaunay triangulation. Note that each new edge swapping (triangulation T^*) insures that the global equiangularity increases $A(T^*) > A(T)$. Because the triangulation is of finite dimension, this guarantees that the process will terminate in a finite number of steps.

Iterative Algorithm: Triangulation via Lawson's Algorithm

```
swapedge = true
While(swapedge)do
  swapedge = false
  Do (*all interior edges*)
    If (*adjacent triangles form convex quad*)then
      *Swap diagonal to form* $T^*$.
      If (*optimization criteria satisfied*)then
        $T = T^*$
        swapedge = true
      EndIf
    EndIf
  EndDo
EndWhile
```

When Lawson's algorithm is used for constructing Delaunay triangulations, the test for quadrilateral convexity is not needed. It can be shown that nonconvex quadrilaterals formed from triangle pairs *never* violate the circumcircle test. When more general optimization criteria is used (discussed later), the convex check must be performed.

3.4 Other 2-D Triangulation Algorithms

In this section, other algorithms which do not necessarily produce Delaunay triangulations are explored.

The MinMax Triangulation

As Babuška and Aziz [22] point out, from the point of view of finite elements the MaxMin (Delaunay) triangulation is not essential. What is essential is that no angle be too close to 180°. In other words, triangulations which minimize the maximum angle are more desirable. These triangulations are referred to as MinMax triangulations. One way to generate a 2-D MinMax triangulations is via Lawson's edge swapping algorithm. In the case, the diagonal position for convex pairs of triangles is chosen which minimizes the maximum interior angle for both triangles. The algorithm is guaranteed to converge in a finite number of steps using arguments similar to Delaunay triangulation. Figures 3.14 and 3.15 present a Delaunay (MaxMin) and MinMax triangulation for 100 random points.

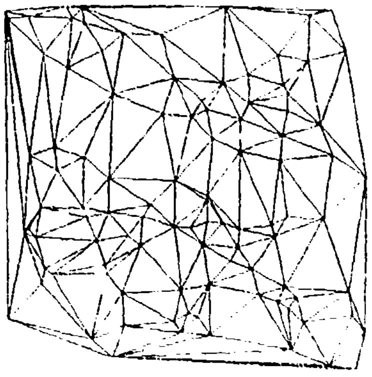


Figure 3.14 Delaunay Triangulation.

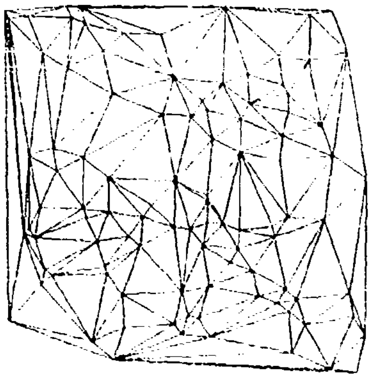


Figure 3.15 MinMax Triangulation.

Note that application of local MinMax optimization via Lawson's algorithm may only result in a mesh which is *locally* optimal and not necessarily at a global minimum. Attaining a globally optimal MinMax triangulation is a much more difficult task. The best algorithm to present date (Edelsbrunner, Tan, and Waupotitsch [23]) has a high complexity of $O(n^2 \log n)$. Wiltberger [24] has implemented a version of the Green and Sibson algorithm [17] which has been modified to produce locally optimal MinMax triangulations using incremental insertion and local edge swapping. The algorithm is implemented using recursive programming with complete forward and backward propagation (contrast figures 3.16 and 3.9). This is a necessary step to produce locally optimized meshes.

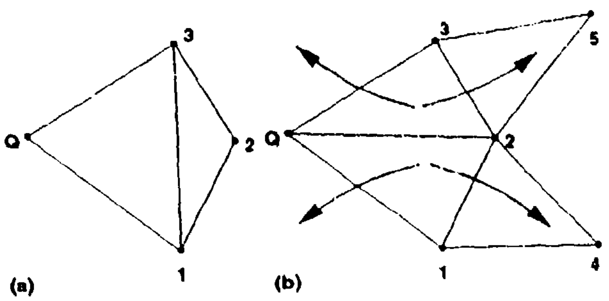


Figure 3.16 Edge swapping with forward and backward propagation in Wiltberger algorithm.

The MinMax triangulation has proven to be very useful in CFD. Figure 3.17 shows the Delaunay triangulation near the trailing edge region of an airfoil with extreme point clustering.

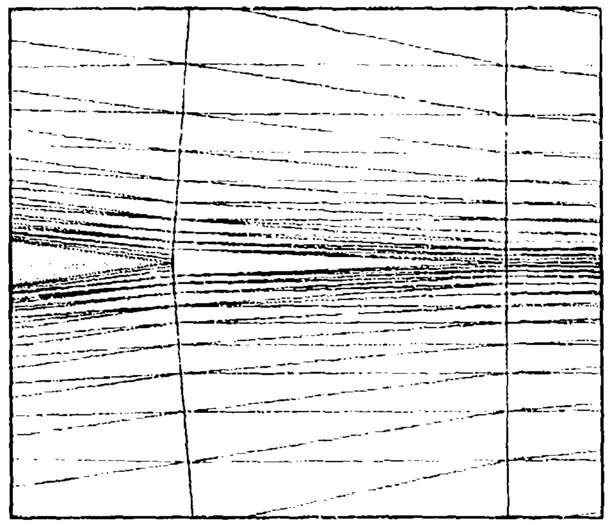


Figure 3.17 Delaunay triangulation near trailing edge of airfoil.

Upon first inspection, the mesh appears flawed near the trailing edge of the airfoil. Further inspection and extreme magnification near the trail edge of the airfoil (figure 3.18) indicates that the grid is a mathematically correct Delaunay triangulation. Because the Delaunay triangulation does not control the maximum angle, the cells near the trailing edge have angles approaching 180°. The presence of nearly collapsed triangles leaves considerable doubt as to the accuracy of any numerical solutions computed in the trailing edge region.

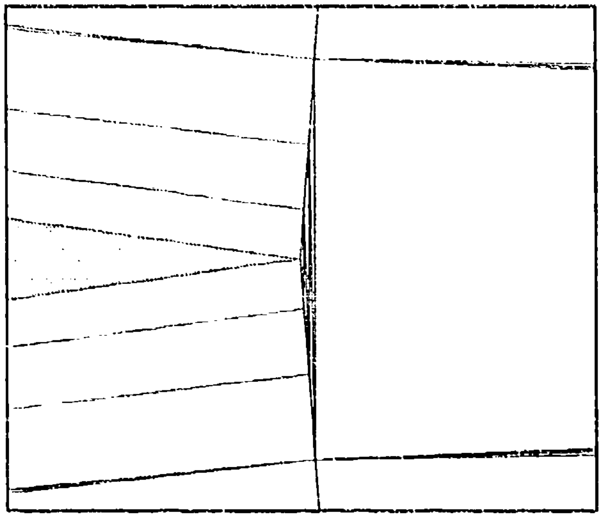


Figure 3.18 Extreme closeup of Delaunay triangulation near trailing edge of airfoil.

Edge swapping based on the MinMax criteria via Lawson's algorithm or incremental insertion using the Wiltberger algorithm produce the desired result as shown in figure 3.19.

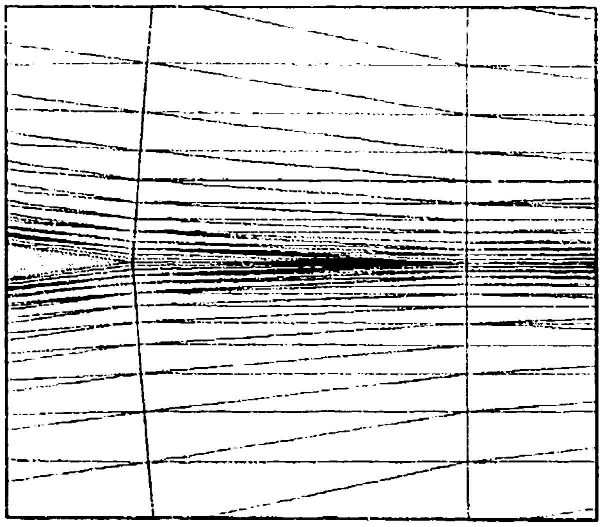


Figure 3.19 MinMax triangulation near trailing edge of airfoil.

The Greedy Triangulation

A greedy method is one that never undoes what it did earlier. The greedy triangulation continually adds edges compatible with the current triangulation (edge crossing not allowed) until the triangulation is complete, i.e. Euler's formula is satisfied. One objective of a triangulation might be to choose a set of edges with shortest total length. The best that the greedy algorithm can do is adopt a local criterion whereby only the shortest edge available at that moment is considered for addition to the current triangulation. (This does not lead to a triangulation with shortest total length.) Note that greedy triangulation easily accommodates constrained triangulations containing interior boundaries and a nonconvex outer boundary. In this case the boundary edges are simply listed first in the ordering of candidate edges. The entire algorithm is outlined below.

```
Algorithm: Greedy Triangulation
Step 1. Initialize triangulation $T$ as empty.
Step 2. Compute $\binom{n}{2}$ candidate edges.
Step 3. Order pool of candidate edges.
Step 4. Remove current edge $e_s$ from ordered pool.
Step 5. If( $e_s$ does not intersect edges of $T$ ) add $e_s$ to $T$
Step 6. If(Euler's formula not satisfied) go to Step 4.
```

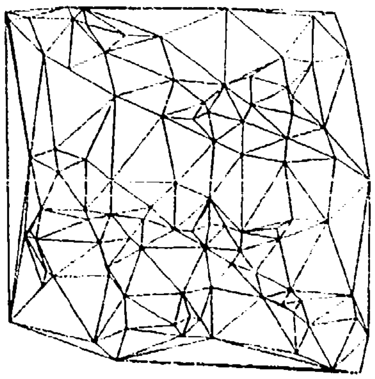


Figure 3.20 Greedy Triangulation.

Figures 3.14 and 3.20 contrast the Delaunay and greedy algorithm. The lack of angle control is easily seen in the greedy triangulation. The greedy algorithm suffers from both high running time as well as storage. In fact a naive implementation of Step 5. leads to an algorithm with $O(N^3)$ complexity. Efficient implementation techniques are given in Gilbert [25] with the result that the complexity can be reduced to $O(N^2 \log N)$ with $O(N^2)$ storage.

Data Dependent Triangulation

Unlike mesh adaptation, a data dependent triangulation assumes that the number and position of vertices is fixed and unchanging. Of all possible triangulations of these vertices, the goal is to find the best triangulation under data dependent constraints. In Nira, Levin, and Rippa [26], they consider several data dependent constraints together with piecewise linear interpolation. In order to determine if a new mesh is "better" than a previous one, a local cost function is defined for each interior edge. Two choices which prove to be particularly effective are the JND (Jump in Normal Derivatives) and the ABN (Angle Between Normals). Using their notation, consider an interior edge with adjacent triangles T_1 and T_2. Let $P(x,y)_1$ and $P(x,y)_2$ be the linear interpolation polynomials in T_1 and T_2 respectively:

$$P_1(x,y) = a_1x + b_1y + c_1$$

$$P_2(x,y) = a_2x + b_2y + c_2$$

The JND cost function measures the jump in normal derivatives of P_1 and P_2 across a common edge with normal components n_x and n_y.

$$s(f_T, e) = |n_x(a_1 - a_2) + n_y(b_1 - b_2)|,$$

(JND cost function)

The ABN measures the acute angle between the two normals formed from the two planes P_1 and P_2. Again using the notation of [26]:

$$s(f_T, e) = \theta = \cos^{-1}(A)$$

$$A = \frac{a_1a_2 + b_1b_2 + 1}{\sqrt{(a_1^2 + b_1^2 + 1)(a_2^2 + b_2^2 + 1)}},$$

(ABN cost function)

The next step is to construct a global measure of these cost functions. This measure is required to decrease for each legal edge swapping. This insures that the edge swapping process terminates. The simplest measures are the l_1 and l_2 norms:

$$R_1(f_T) = \sum_{edges} |s(f_T, e)|$$

$$R_2(f_T) = \sum_{edges} s(f_T, e)^2$$

Recall that a Delaunay triangulation would result if the cost function is chosen which maximizes the minimum angle between adjacent triangles (Lawson's algorithm). Although it would be desirable to obtain a global optimum for all cost functions, this could be very costly in many cases. An alternate strategy is to abandon the pursuit of a globally optimal triangulation in favor of a locally optimal triangulation. Once again Lawson's algorithm is used. Note that in using Lawson's algorithm, we require that the global measure decrease at each edge swap. This is not as simple as before since each edge swap can have an effect on other surrounding edge cost functions. Nevertheless, this domain of influence is very small and easily found.

Iterative Algorithm: Data Dependent Triangulation via Modified Lawson's Algorithm

```
swapedge = true
While(swapedge)do
  swapedge = false
  Do (all interior edges)
    If (adjacent triangles
        form convex quadrilateral)then
      Swap diagonal to form $T^*$.
      If ($R(f_{T^*}) < R(f_T)$)then
        $T = T^*$
        swapedge = true
      EndIf
    EndIf
  EndDo
EndWhile
```

Edge swapping only occurs when $R(f_{T^*}) < R(f_T)$ which guarantees that the method terminates in a finite number of steps. Figures 3.14 and 3.21 plot the Delaunay triangulation of 100 random vertices in a unit square and piecewise linear contours of $(1 + \tanh(9y - 9x))/9$ on this mesh. The exact solution consists of straight line contours with unit slope.

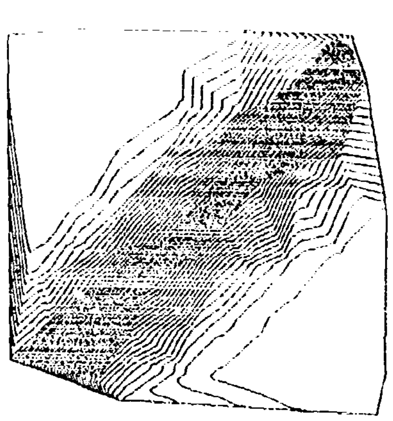


Figure 3.21 Piecewise Linear Interpolation of $(1 + \tanh(9y - 9x))/9$.

In figures 3.22 and 3.23 the data dependent triangulation and solution contours using the JND criteria and l_1 measure suggested in [26] are plotted.

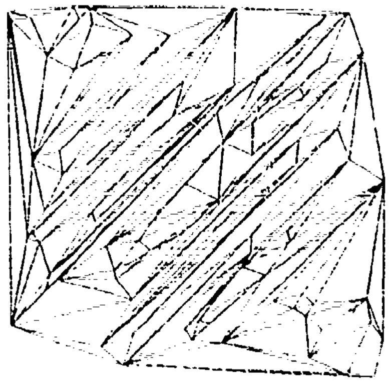


Figure 3.22. Data Dependent Triangulation.

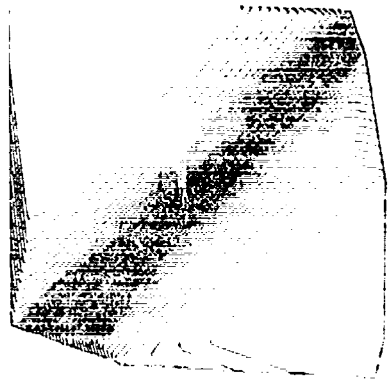


Figure 3.23 Piecewise Linear Interpolation of $(1 + \tanh(9y - 9x))/9$.

Note that the triangulations obtained from this method are not globally optimal and highly dependent on the order in which edges are accessed. Several possible ordering strategies are mentioned in [27].

3.5 2-D Steiner Triangulations

Definition: A Steiner triangulation is any triangulation that adds additional sites to an existing triangulation to improve some measure of grid quality.

Technically speaking, the method of advancing front grid generation discussed by Professors Morgan and Löhner in these notes would be a special type of Steiner triangulation. The insertion algorithms described earlier also provide a simple mechanism for generating Steiner triangulations. Holmes [28] demonstrated the feasibility of inserting sites at circumcenters of Delaunay triangles into an existing 2-D triangulation to improve measures of grid quality. This has the desired effect of placing the new site in a position that guarantees that no other site in the triangulation can lie closer that the radius of the circumcircle, see figure 3.24. In a loose sense, the new site is placed as far away from other nearby sites as conservatively possible.

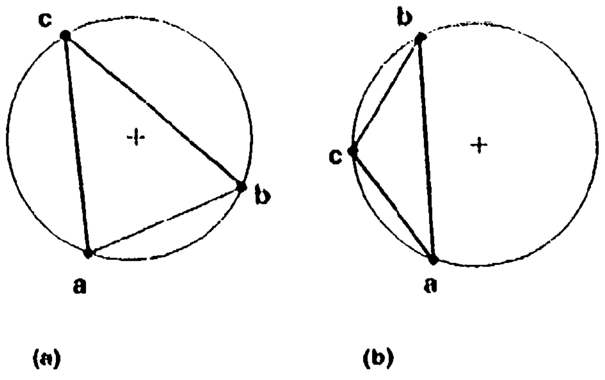


Figure 3.24 Inserting site at circumcenter of acute and obtuse triangles.

Warren *et al* [29] and Anderson [30] further demonstrated the utility of this type of Steiner triangulation in the generation and adaptive refinement of 2-D meshes. The algorithm developed by Wiltberger [24] also permits Steiner triangulations based on either MinMax or MaxMin (Delaunay) insertion. Only in the latter case is the insertion at triangle circumcenters truly justifiable. The paragraphs below give an expanded discussion of 2-D Steiner triangulation.

Steiner Grid Generation

The 2-D Steiner point grid generation algorithm described in [28,29,30] consists of the following steps. The first step is the Delaunay triangulation of the boundary data. Usually three or four points are placed in the far field with convex hull enclosing all the boundary points. Starting with a triangulation of these points, sites corresponding to boundary curves are incrementally inserted using Watson's algorithm in [28,29,30] and Green and Sibson's algorithm in [17] as shown in figure 3.25. The initial triangulation does not guarantee that all boundary edges are members of the triangulation. This can be remedied in a variety of ways. One technique adds additional points to the triangulation so as to guarantee that the resulting Delaunay triangulation contains all the desired boundary edges, see reference [16]. Another approach performs local edge swapping so as to produce a *constrained* Delaunay triangulation which guarantees that all boundary edges are actual edges of the triangulation.

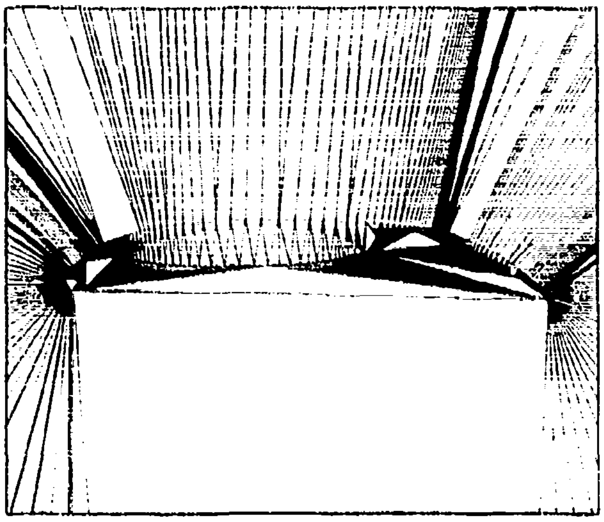


Figure 3.25 Initial triangulation of boundary points.

In either event, the boundary edges are marked so that they cannot be removed as the triangulation is refined. The algorithms described in [28,29,30] interrogate triangles in an arbitrary order (this makes the triangulation nonunique). The user must specify some measure of quality for triangle refinement (aspect ratio, area, containment circle radius, for example) and a threshold value for the measure. If a triangle fails to meet the threshold value, the triangulation is refined by placing a new site at the circumcenter of the failed triangle via Watson's algorithm. Some care must be taken to insure that measures are chosen which are guaranteed to be reduced when the refinement takes place. Using thresholding in this way does not give the user direct control over the actual number of triangle generated in the process of Steiner refinement. Wiltberger takes a different approach by maintaining a *dynamic heap* data structure of the quality measure. (Heap structures are a very efficient way of keeping a sorted list of entries with insertion and query time $O(\log N)$ for N entries.) The triangle with the largest value of the specified measure will be located at the top of the heap at all times during the triangulation. This makes implementation of a Steiner triangulation which *minimizes the maximum value* of the measure very efficient (and unique). In this implementation, the user can either specify the number of triangles to be generated or a threshold value of the measure. Note that multiple measures can be refined lexicographically. Figure 3.26 shows a Steiner triangulation using the Wiltberger algorithm with MaxMin insertion and refinement based on maximum aspect ratio.

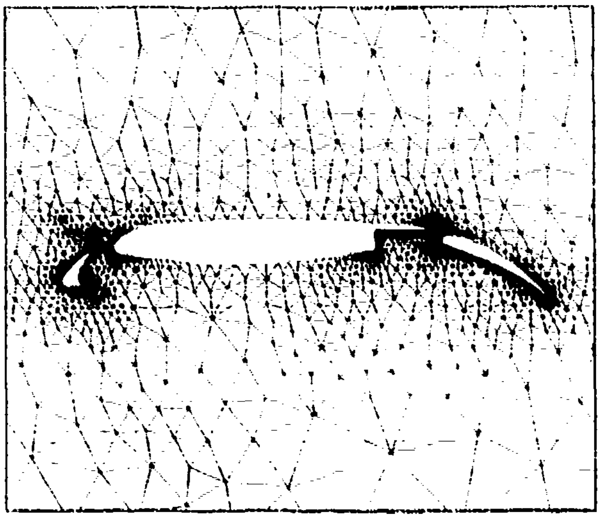


Figure 3.26 Steiner triangulation with sites inserted at circumcenters to reduce maximum cell aspect ratio.

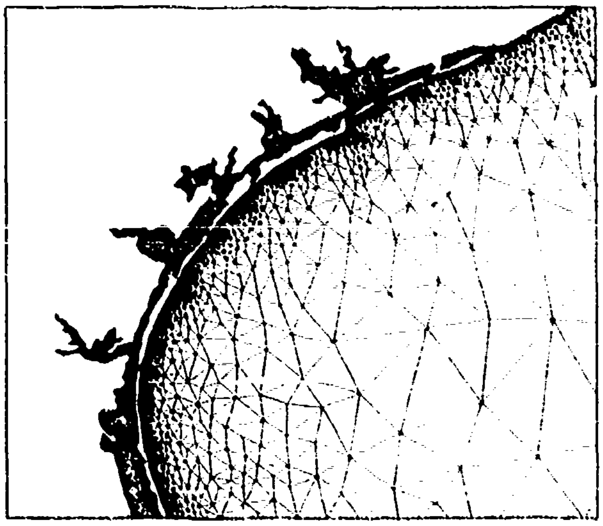


Figure 3.27 Steiner triangulation of Texas coast and the Gulf of Mexico.

This triangulation has proven to be very flexible. For instance, figure 3.27 shows a Steiner triangulation of the Texas coast and Gulf of Mexico.

3.6 Three-Dimensional Triangulations

The Delaunay triangulation extends naturally into three dimensions as the geometric dual of the 3-D Voronoi diagram. The Delaunay triangulation in 3-D can be characterized as the unique triangulation such that the circumsphere passing through the four vertices of any tetrahedron must not contain any other point in the triangulation. As in the 2-D case, the 3-D Delaunay triangulation has the property that it minimizes the maximum containment sphere (globally but not locally). In two dimensions, it can be shown that a mesh entirely comprised of acute triangles is automatically Delaunay. To prove this, consider an adjacent triangle pair forming a quadrilateral. By swapping the position of the diagonal it is easily shown that the minimum angle always increases. Rajan [12] shows the natural extension of this idea to three or more space dimensions. He defines a "self-centered" simplex in $\mathbf{R}^d$ to be a simplex which has the circumcenter of its circumsphere interior to the simplex. In two dimensions, acute triangles are self-centered and obtuse triangles are not. Rajan shows that a triangulation entirely composed of self-centered simplices in $\mathbf{R}^d$ is automatically Delaunay.

3.6a 3-D Bowyer and Watson Algorithms

The algorithms of Bowyer [14] and Watson [15] extend naturally to three dimensions with estimated complexities of $O(N^{5/3})$ and $O(N^{4/3})$ for N randomly distributed vertices. They do not give worst case estimates. It should be noted that in three dimensions, Klee [20] shows that the maximum number of tetrahedra which can be generated from N vertices is $O(N^2)$. Thus an optimal worst case complexity would be a least $O(N^2)$. Under normal conditions this worst case scenario is rarely encountered. Baker [16] reports more realistic actual run times for Watson's algorithm.

3.6b 3-D Edge Swapping Algorithms

Until most recently, the algorithm of Green and Sibson based on edge swapping was thought not to be extendable to three dimensions because it was unclear how to generalize the concept of edge swapping to three or more dimensions. In 1986, Lawson published a paper [21] in which he proved the fundamental combinatorial result:

Theorem: (Lawson, 1986) The convex hull of $d+2$ points in $\mathbf{R}^d$ can be triangulated in at most 2 ways.

Joe [31,32], and Rajan [12] have constructed algorithms based on this theorem. In joint work with A. Gandhi [33], we independently constructed an incremental Delaunay triangulation algorithm based on Lawson's theorem. The remainder of this section will review our algorithm and the basic ideas behind 3-D edge swapping algorithms.

It is useful to develop a taxonomy of possible configurations addressed by Lawson's theorem. Figure 3.28 shows configurations in 3-D of five points which can be triangulated in only one way and hence no change is possible. We call these arrangements "unswappable". Figure 3.29 shows configurations which allow two ways of triangulation. It is possible to flip between the two possible triangulations and we call these arrangements "swappable".

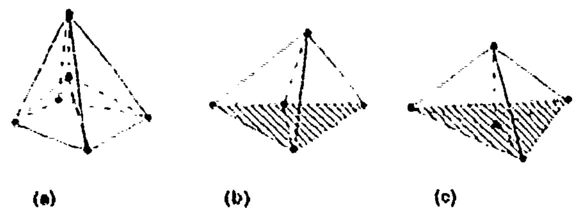


Figure 3.28 Generic nonswappable configurations of 5 points. Shaded region denotes planar surface.

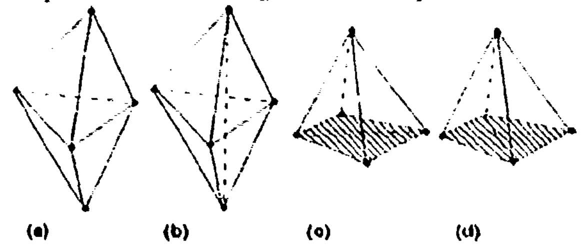


Figure 3.29 Generic swappable configurations of 5 points. Shaded region denotes planar surface.

There are two arrangements that allow two triangulations. Figures 3.29(a) and 3.29(b) show the subclass of companion triangulations that can be transformed from one type to another thereby changing the number of tetrahedra from 2 to 3 or vice-versa. Figures 3.29(c) and 3.29(d) show the other subclass of configurations that can be transformed from one type to another while keeping constant the number of tetrahedra (2). These figures reveal an important difference between the two and three-dimensional algorithms. *The number of tetrahedrons involved in the swapping operation need not be constant.*

The 3-D edge swapping algorithm is based on flipping between the two ways of triangulating the configurations in figure 3.29. One good way of finding all sets of five points in the mesh is to loop through all the faces in the mesh and considering the five points that make up the two adjoining tetrahedra for that face. Below we present the facewise edge swapping primitive.

Primitive: EDGE_SWAP(face)

Let C = { Set of tetrahedra made from the 5 nodes of the two adjoining tetrahedra }

```
If(shape(C) = convex)then
   Let $T$ = current triangulation
   Let $T^*$ = alternate triangulation (if it exists)
   If( Quality($T^*$) > Quality($T$))then
      [Edge Swap $T$ into $T^*$]
   Endif
Endif
```

The first step is to find *all* the tetrahedra that are described by the five nodes of the two tetrahedra adjacent to the face in question. There is a maximum of four tetrahedra that can be built from five nodes. Since any two tetrahedra made from the same five points will have to share three points (i.e., a face) it is sufficient to only look at the four neighboring tetrahedra of any of the two tetrahedra already known. This constitutes a linear time algorithm for finding all the non-overlapping tetrahedra made from the five points.

If these tetrahedra form a convex shape, then the configuration is described by one of the configurations in figures 3.28 and 3.29 (configurations of the convex hull) and edge swapping is permitted. If, for example, only two of the three tetrahedra in figure 3.29(b) were present, the two tetrahedra will form a concave shape. Obviously, edge swapping concave shapes is not possible without possibly creating overlapping tetrahedra in the mesh. For swappable configurations, a check is performed to see if the local mesh quality measure (discussed further below) will improve by edge swapping into the alternate triangulation. If it does, the swap is performed; otherwise the triangulation is unchanged. This technique offers a distinct advantage over others (Bowyer's or Watson's algorithm) in that it allows the use of any arbitrary mesh quality measure.

Computational Aspects of 3-D Edge Swapping

Before proceding further, it is useful to discuss the computational aspects of some of the operations needed for the edge swapping algorithm.

◆ Determining Convexity: This operation tests whether the shape formed by the tetrahedron T_1 and T_2 is convex. Let the vertices of the tetrahedra be numbered $T_1 = (1,2,3,4)$ and $T_2 = (1,2,3,5)$ i.e., $(1,2,3)$ is the face shared by T_1 and T_2 and nodes 4 and 5 are at the two ends of T_1 and T_2 respectively. We make use of the notion of barycentric coordinates to perform the convexity test. The $b_{1,2,3,4}$ satisfying

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix}$$

are called the barycentric coordinates of node 5. They indicate the position of 5 in relation to the nodes of tetrahedron T_1. For each s, the sign of b_s indicates the position of 5 relative to the plane H_s passing through the triangular face opposite node s. Thus $b_s = 0$ when 5 is in H_s, $b_s > 0$ when 5 is on the same side of H_s as node s, and $b_s < 0$ when 5 is on the opposite side of H_s from node s. Clearly, $b_{1,2,3,4} > 0$ if 5 lies inside tetrahedron $(1,2,3,4)$. If we imagine a cone formed by planes $H_{1,2,3}$ of T_1, then T_1 and T_2 would form a convex shape if and only if node 5 lies in the cone on the side opposite from node 4 (figure 3.30).

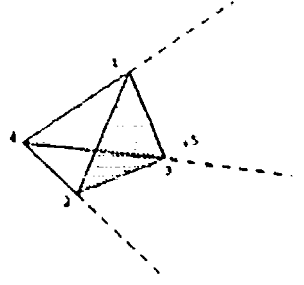


Figure 3.30 Convexity cone for node 4.

The conditions to satisfy this requirement are $b_4 < 0$ and $b_{1,2,3} > 0$. In order to test if three cells form a convex shape, the nodes are renumbered as if the three cell configuration were edge swapped into the corresponding two cell configuration for purposes of the barycentric test. For example, consider a three cell configuration with numbering $(1,2,3,4)$, $(2,3,4,5)$, and $(1,2,4,5)$, then the corresponding two cell configuration would have $(1,3,5)$ as the common face and nodes 2, and 4 at the ends of the two tetrahedra. Notice that if the three tetrahedra formed a convex shape, then it would be possible to edge swap it to a convex two tetrahedra configuration. If the three cells formed a concave shape, however, the transformed two cell triangulation would contain overlapping tetrahedra which the test above would label as concave.

Using Cramer's rule to solve the $Ax = b$ problem posed above requires computing determinants of the form

$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{vmatrix}$$

five times. An optimization is possible by exploiting the property of determinants that subtracting one row or column from another leaves the determinant unchanged. If we subtract the first column from the rest, we simplify the above 4x4 determinant into a 3x3 one.

$$\begin{vmatrix} 1 & 0 & 0 & 0 \\ a_{11} & a_{12}-a_{11} & a_{13}-a_{11} & a_{14}-a_{11} \\ a_{21} & a_{22}-a_{21} & a_{23}-a_{21} & a_{24}-a_{21} \\ a_{31} & a_{32}-a_{31} & a_{33}-a_{31} & a_{34}-a_{31} \end{vmatrix} = \begin{vmatrix} a_{12}-a_{11} & a_{13}-a_{11} & a_{14}-a_{11} \\ a_{22}-a_{21} & a_{23}-a_{21} & a_{24}-a_{21} \\ a_{32}-a_{31} & a_{33}-a_{31} & a_{34}-a_{31} \end{vmatrix}$$

• Delaunay Circumsphere test: The 3-D Delaunay triangulation is defined as the unique triangulation such that the circumsphere of any tetrahedron contains no other point in the mesh. To determine where point E lies in relation to the circumsphere of tetrahedron (A, B, C, D), denoted by($\bigcirc$ $ABCD$), we use the *InSphere* primitive :

$$InSphere(E)\begin{cases} <0 & \text{if } E \text{ is inside } \bigcirc\ ABCD \\ =0 & \text{if } E \text{ is on } \bigcirc\ ABCD \\ >0 & \text{if } E \text{ is outside } \bigcirc\ ABCD \end{cases}$$

where *InSphere* is computed from the following determinant:

InSphere(E)

$$= \begin{vmatrix} 1 & 1 & 1 & 1 & 1 \\ x_A & x_B & x_C & x_D & x_E \\ y_A & y_B & y_C & y_D & y_E \\ z_A & z_B & z_C & z_D & z_E \\ w_A^2 & w_B^2 & w_C^2 & w_D^2 & w_E^2 \end{vmatrix} \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_A & x_B & x_C & x_D \\ y_A & y_B & y_C & y_D \\ z_A & z_B & z_C & z_D \end{vmatrix}$$

$$= \begin{vmatrix} x_B-x_A & x_C-x_A & x_D-x_A & x_E-x_A \\ y_B-y_A & y_C-y_A & y_D-y_A & y_E-y_A \\ z_B-z_A & z_C-z_A & z_D-z_A & z_E-z_A \\ w_B^2-w_A^2 & w_C^2-w_A^2 & w_D^2-w_A^2 & w_E^2-w_A^2 \end{vmatrix}$$

$$\begin{vmatrix} x_B-x_A & x_C-x_A & x_D-x_A \\ y_B-y_A & y_C-y_A & y_D-y_A \\ z_B-z_A & z_C-z_A & z_D-z_A \end{vmatrix}$$

and $w_P^2 = x_P^2 + y_P^2 + z_P^2$

The first determinant is the 3-D extension of Guibas' *InCircle* primitive [1]. It represents the volume of a pentatope whose vertices are the points A, B, C, D, E projected onto the 4-D paraboloid $(x^2 + y^2 + z^2)$. (A pentatope is the simplest polytope in 4-D just as a tetrahedron is the simplest polytope in 3-D and a triangle in 2-D. A pentatope can be constructed by joining the tetrahedron to a fifth point outside its 3-space.) The coordinates in 3-space of these five points remain unchanged; they simply acquire a value in their fourth coordinate equal to the square of their distances from the origin. The volume of this polytope is positive if point E lies outside $\bigcirc$ $ABCD$ and negative if point E lies inside $\bigcirc$ $ABCD$, provided that tetrahedron (A, B, C, D) has a positive volume (as given by the second determinant). The determinant is degenerate if point E lies exactly on $\bigcirc$ $ABCD$.

This test is motivated by the observation in 3-D that the intersection of a cylinder and a unit paraboloid is an ellipse lying in a plane (figure 3.31). So, any four co-circular points in 2-D will project to four co-planar points and the volume of the tetrahedra made from these four co-planar points will be zero. The paraboloid is a surface that is convex upward and the points interior to a circle get projected to the paraboloid below the intersection plane and the points exterior to it get projected above the the intersection plane. If a point lies outside the circumcircle of three other points, the tetrahedron made from these four points will have positive volume provided the three points were ordered in a counter-clockwise fashion. The volume will be negative if the point lies inside the circumcircle.

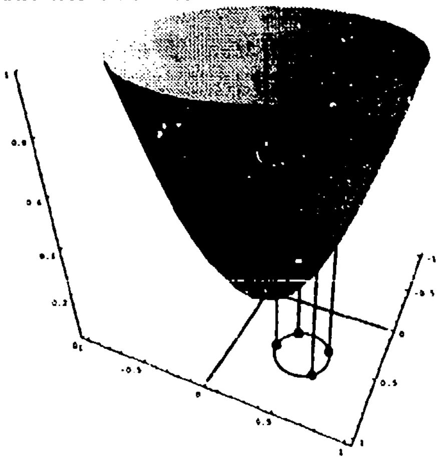


Figure 3.31 Projection of cocircular points onto unit paraboloid.

The second determinant is the 3-D extension of Guibas' *CCW* (counter clock-wise) primitive [1] which computes the volume of tetrahedron (A,B,C,D). Thus the *InSphere* primitive works irrespective of how the points A, B, C, and D are ordered. If it can be guaranteed that all the tetrahedra in a mesh have their vertices ordered to have positive volumes then the need to compute the second determinant is eliminated. The *InSphere* primitive becomes ill-behaved when the points A, B, C, and D all lie nearly on a plane because the position of the circumsphere with respect to the points (i.e., whether the circumsphere is above or below the plane) becomes very sensitive to small perturbations in the coordinates of the five points.

3-D Mesh Optimization

The 3-D edge swapping algorithm can be used to optimize existing triangulations. In fact, there is no way to triangulate a given set of points based on the minmax or maxmin of the face angles directly. An alternative is to start with an existing triangulation and optimize it. This requires that we cycle through all the faces in a mesh and apply the edge swapping procedure at each step. This process is continued until no more swaps are possible.

Algorithm: Three-dimensional mesh optimization.

```
while (swaps occurred in the last cycle over faces)
  for all faces
    EDGE_SWAP (face)
  endfor
endwhile
```

In the following paragraphs, we discuss a few swap-criteria and examine the meshes they produce.

Global Edge Swapping

The *InSphere* criteria is binary in the sense that either the triangulation of a set of five points satisfies the criteria or it does not. It can also be shown that of the two ways to triangulate a set of five points, if one way fails the *InSphere* criteria, then the other one will pass and vice-versa. Cases 1 and 3 in figure 3.15 and cases 1, 3, and 5 in figure 3.16 will always pass the *InSphere* criteria.

The Delaunay triangulation is unique for a given set of points. Lawson also noticed the relation between local and global properties of the Delaunay *InSphere* criteria: a triangulation is Delaunay if and only if the triangulations of sets of five points corresponding to all the interior faces in the mesh satisfy the *InSphere* criteria. This means that if every face satisfies the Delaunay criteria, then the whole mesh must be a Delaunay triangulation.

Joe [32] has proven, however, that processing faces in an arbitrary way may result in getting stuck in local optima. This is an important difference between two and three dimensional combinatorial edge swapping.

3-D MinMax and MaxMin Triangulations

The edge swapping algorithm can be applied locally to produce a triangulation that minimizes the maximum face angle. In 2-D, the edge swapping algorithm (working with edge angles) gets stuck in local minima and depending on the order in which the edges were traversed, different local minima are reached. In practice, the local minima all seem very close to the global minimum which makes edge swapping a practical way to get a nearly optimal MinMax triangulation. We observe that in 3-D as well, there are many local minima and the order of face traversal determines which one is found. It is hard to determine how far these local minima are from the global minimum but we believe that edge swapping is a practical way to get nearly optimal MinMax meshes.

Lawson has shown that in 2-D, Delaunay triangulations have the property that the minimum edge angle is maximized (i.e., MaxMin triangulation). So in 2-D, the MaxMin triangulation is unique and the edge swapping algorithm will converge to it. In 3-D, however, the Delaunay triangulation is not the same as MaxMin triangulation and the edge swapping algorithm working with the MaxMin criteria has the same property of getting stuck in local minima as the MinMax. Again, it is hard to judge how close the local minima are from the global minimum but we still conclude that edge swapping is a fairly efficient technique for the construction of MaxMin triangulations.

3-D Minimum Edge Triangulation

Another mesh of interest is the minimum edge triangulation. Since finite-volume flow solvers work edge-wise, it is beneficial to reduce the number of edges in a mesh. This is easily accomplished by edge swapping such that we always swap from case 3.29a to case 3.29b. Each time this operation is performed, one edge and one tetrahedron are removed from the mesh. Again, different meshes will be produced depending upon how faces are traversed and the final mesh may only be at a local minimum.

Incremental Delaunay Triangulation

The edge swapping algorithm provides an effective way for inserting a point into an existing triangulation. Simply find the tetrahedra into which the point is to be inserted and test its faces according to the circumsphere criteria to determine if edge swapping should take place. If a set of 5 points corresponding to a face is retriangulated, we proceed to test all the outer faces of the new triangulation for swappability and so on. This propagates a front that retriangulates the mesh. It is known that any new face created during the retriangulation is indeed a part of the final mesh as well, and so back-propagation is not required. This may not be true for other mesh quality measures and back propagation then becomes necessary.

Rajan proves that it is possible to find a certain sequence of edge swaps which will guarantee that Delaunay triangulation is recovered when a site is added to an existing Delaunay triangulation. In practice, however, we find that this ordering of edge swaps does not seem to be necessary in order to recover the Delaunay triangulation. In fact, *it is our conjecture that this is always the case.*

This insertion algorithm can be used to adaptively refine meshes. To do this, sites are inserted at the centers of the circumspheres of tetrahedra with large aspect ratios (or other suitable measures). This insertion site does not always lie within the cell T_i marked for refinement. To find the cell in which the new site lies, a walking algorithm is employed. Starting at T_i, barycentrics are computed to determine which face of T_i the new site lies behind. The next step is to traverse to the cell behind that face. This procedure is applied recursively until the cell in which the new site falls within is found. The idea of introducing new sites at the centers of the circumspheres of tetrahedra works well because each new site intro[illegible]ced is equidistant to the 4 points of the large aspect ratio tetrahedra. This produces high quality meshes in 2-D and seems to work well in 3-D.

3.6c 3-D Surface Tr[illegible]gulation

The Wilto[illegible] algorithm has been extended to include the [illegible]gulation of surface patches. Although the [illegible]ept of Dirichlet tessellation is well defined on a [illegible]ooth manifolds using the concept of [illegible]odesic di[illegible]nce, in practice this is too expensi[illegible] Finding geodesic distance is a variation [illegible] at is not easily solved. We have [illegible] [illegible]mpler procedure in which surface grids in 3-D are constructed from rectangular surface patches (assumed at least C^0 smooth) using a generalization of the 2-D Steiner triangulation scheme.

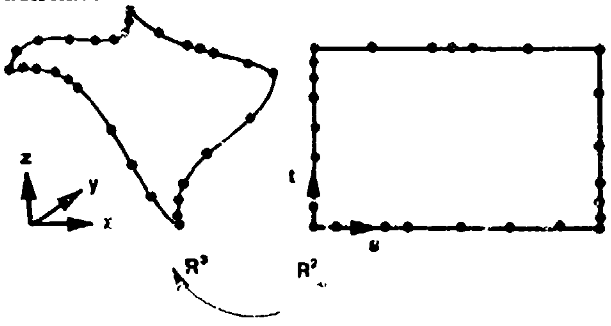


Figure 3.32 Mapping of rectangular patches on (s,t) plane.

Points are first placed on the perimeter of each patch using an adaptive refinement strategy based on absolute error and curvature measures. The surface patches are projected onto the plane, see figure 3.32. Simple stretching of the rectangular patches permits the user to produce preferentially stretched meshes. (This is useful near the leading edge of a wing for example.)

The triangulation takes place in the two dimensional (s,t) plane. The triangulation is adaptively refined using Steiner point insertion to minimize the maximum user specified absolute error and curvature tolerance on each patch. The absolute error is approximated by the perpendicular distance from the triangle centroid (projected back to 3-space) to the true surface as depicted in figure 3.33. The user can further refine based on triangle aspect ratio in the (s,t) plane if desired.

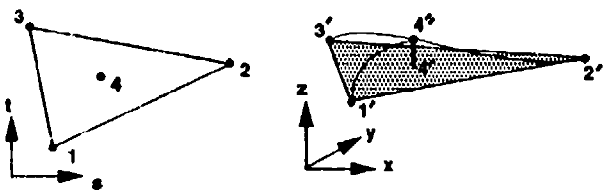


Figure 3.33 Calculation of triangulation absolute error by measurement of distance from face centroid to true surface.

Figure 3.34 shows a typical adaptive surface grid generated using the Steiner triangulation method.

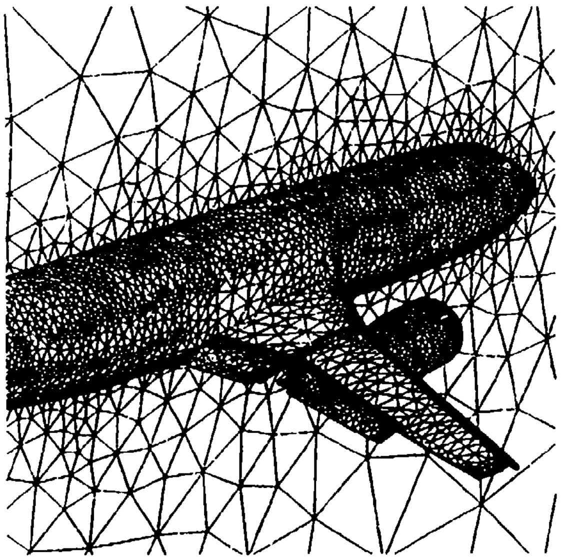


Figure 3.34 Adaptive Steiner triangulation of surface mesh about Boeing 737 with flaps deployed.

4.0 Some Theory Related to Finite-Volume Solvers

4.1 Scalar Conservation Law Equations

For purposes of these notes, we consider numerical methods for solving *conservation law* equations.

Definition: A conservation law asserts that the rate of change of the total amount of a substance with density z in a fixed region Ω is equal to the flux $\mathbf{F}$ of the substance through the boundary $\partial\Omega$.

$$\frac{\partial}{\partial t}\int_{\Omega} z\, da + \int_{\partial\Omega} \mathbf{F}(z)\cdot\mathbf{n}\, dl = 0 \quad \text{(integral form)}$$

The choice of a numerical algorithm used to solve a conservation law equation is often influenced by the form in which the conservation law is presented. A finite-difference practitioner would apply the divergence theorem to the integral form and let the area of Ω shrink to zero thus obtaining the divergence form of the equation.

$$\frac{\partial}{\partial t} z + \nabla\cdot\mathbf{F}(z) = 0 \qquad \text{(divergence form)}$$

The finite-element practitioner constructs the divergence form then multiplies by an arbitrary test function ϕ and integrates by parts.

$$\frac{\partial}{\partial t}\int_{\Omega}\phi z\, da - \int_{\Omega}\nabla\phi\cdot\mathbf{F}(z)\, da + \int_{\partial\Omega}\phi\mathbf{F}(z)\cdot\mathbf{n}\, dl = 0 \quad \text{(weak form)}$$

Algorithm developers starting from these three forms can produce seemingly different numerical schemes. In reality, the final discretizations are usually very similar. Some differences do appear in the handling of boundary conditions, solution discontinuities, and nonlinearities. When considering flows with discontinuities, the integral form appears advantageous since conservation of fluxes comes for free and the proper jump conditions are assured. At discontinuities, the divergence form of the equations implies satisfaction in the sense of distribution theory. Consequently, at discontinuities special care is needed to construct finite difference schemes which produce physically relevant solutions. Because the test functions have compact support, the weak form of the equations also guarantees satisfaction of the jump conditions over the extent of the support. The divergence form of the equations is rarely used in the discretization of conservation law equations on unstructured meshes because of the difficulty in ensuring conservation. On the other hand, the integral and weak forms are both used extensively in numerical modeling of conservation laws on unstructured meshes. In the next section, the simplest of numerical schemes based on integral and weak forms of the conservation law are compared to illustrate their similarities. These schemes can be viewed as the "central-difference" counterparts on unstructured grids. For advection dominated flows, these algorithms are inadequate and additional terms must be added. This topic is undertaken in detail in future sections.

4.2 Comparison of Finite-Volume and Galerkin Finite-Element Methods

Although the integral and weak forms of the equations appear to be quite different, numerical schemes based on these forms often produce identical discretizations. To demonstrate this point, consider the Galerkin discretization (with linear elements) of a general model advection-diffusion equation ($\mu > 0$):

$$\frac{\partial}{\partial t} z + \nabla\cdot\mathbf{F}(z) = \nabla\cdot\mu\nabla z$$

Multiplying by a test function ϕ and integrating by parts over the region Ω produces the weak form of the equation.

$$\frac{\partial}{\partial t}\int_{\Omega}\phi z\, da - \int_{\Omega}\nabla\phi\cdot\mathbf{F}(z)\, da + \int_{\partial\Omega}\phi\,\mathbf{F}(z)\cdot\mathbf{n}\, dl = -\int_{\Omega}\mu\nabla\phi\cdot\nabla z\, da + \int_{\partial\Omega}\mu\phi\nabla z\cdot\mathbf{n}\, dl \tag{4.0}$$

In the finite-element method, the entire domain is first divided into smaller elements. In this case, the elements are triangles T_j, such that $\Omega = \cup T_j$, $T_j \cap T_l = \emptyset$, $l \neq j$. In Fig. 4.1a we show a representative vertex with adjacent neighbors. (To simplify the discussion in the remainder of these notes, we adopt the convention that the index "j" refers to a global index of a mesh whereas the index "i" always refers to a local index.) The linear variation of the solution in each triangle T_j can be expressed in terms of the three local nodal values of the solution, $z^h_{T_j,i}$, $i = 1,2,3$, and three element shape functions n_i, $i = 1,2,3$.

$$z^h(x,y)_{T_j} = \sum_{i=1}^{3} n_i(x,y)\, z^h_{T_j,i} \quad \text{(local representation)}$$

Each element shape function n_i can be interpreted as a piecewise linear surface which takes on a unit value at v_i and vanishes at the other two vertices of the triangle as well as everywhere outside the triangle. The solution can also be expressed globally in terms of nodal values of the solution and global shape functions.

$$z^h(x,y) = \sum_{nodes} N_j(x,y)\; z_j^h$$

(global representation)

In this form, the global shape functions are piecewise linear pyramids which are formed from the union of all local shape functions with have unit value at v_j. These global shape functions also enjoy compact support, i.e. they vanish outside the region Ω_j formed from the union of all triangles incident to v_j. A global shape function for vertex v_j is shown in Fig. 4.1b.

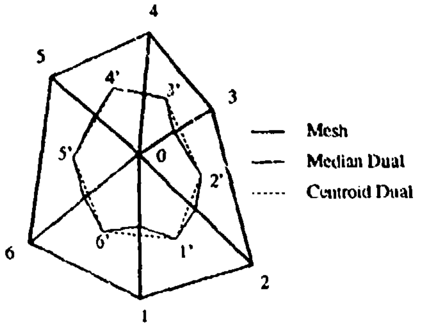


Figure 4.1a Local mesh with centroid and median duals.

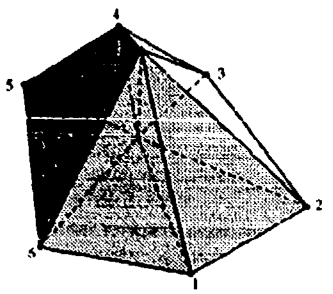


Figure 4.1b Global shape function for vertex v_0 (not labeled).

The Galerkin finite-element method assumes that the class of test functions is identical to the class of functions approximating the solution. The simplest test functions of this sort are the individual shape functions. To obtain a Galerkin discretization for a typical vertex v_j, simply set $\phi^h = N_j$ and evaluate (4.0) in Ω_j. Since ϕ vanishes on $\partial\Omega_j$ equation (4.0) simplifies to the following form:

$$\frac{\partial}{\partial t}\int_{\Omega_j} \phi^h z^h \, da - \int_{\Omega_j} \nabla\phi^h \cdot \mathbf{F}(z^h)\, da = -\int_{\Omega_j} \mu \nabla\phi^h \cdot \nabla z^h \, da \qquad (4.1)$$

Before evaluating equation (4.1), it is useful to introduce more notation concerning the geometry of figure 4.1a. Figure 4.2 depicts the index and normal convention which will be used throughout these notes. The triangle with vertices 0, i, and $i+1$ is denoted as $T_{i+1/2}$. This index convention will be used for other quantities such as areas and gradients which are computed in $T_{i+1/2}$.

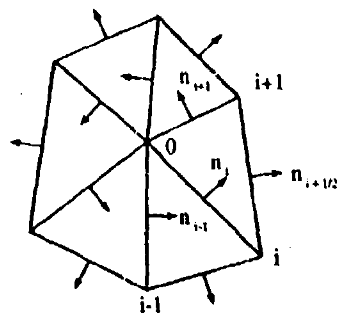


Figure 4.2 Vertex v_0 and adjacent neighbors.

It is convenient to define normals, $\vec{\mathbf{n}}$, for straight edges which are scaled by the length of the edge. Using this notation, a simple formula exists for the gradient of the numerical solution in a triangle $T_{i+1/2}$.

$$\nabla z^h_{i+1/2} = \frac{-1}{2A_{i+1/2}} \left(z_0^h \vec{\mathbf{n}}_{i+1/2} + z_i^h \vec{\mathbf{n}}_{i+1} - z_{i+1}^h \vec{\mathbf{n}}_i \right) \qquad (4.2)$$

The gradient of the test function in each triangle takes a similar form (replace z by ϕ in the previous formula with $\phi_0 = 1, \phi_i = 0, \phi_{i+1} = 0$).

$$\nabla\phi_{i+1/2} = \frac{-1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} \qquad (4.3)$$

The discrete form of eqn. (4.0) is now written as

$$\frac{\partial}{\partial t}\int_{\Omega_j} \phi z^h \, da + \sum_{i=1}^{d(v_0)} \frac{\vec{\mathbf{n}}_{i+1/2}}{2A_{i+1/2}} \cdot \int_{T_{i+1/2}} \mathbf{F}(z^h)\, da = \sum_{i=1}^{d(v_0)} \frac{\vec{\mathbf{n}}_{i+1/2}}{2A_{i+1/2}} \cdot \int_{T_{i+1/2}} \mu \nabla z^h \, da \qquad (4.4)$$

The flux integral can be evaluated by exact integration (when possible) or numerical quadrature. In this case, the latter is assumed.

$$\int_{T_{i+1/2}} \mathbf{F}(z^h)\, da = \frac{A_{i+1/2}}{3} \left(\mathbf{F}(z_0^h) + \mathbf{F}(z_i^h) + \mathbf{F}(z_{i+1}^h) \right) \qquad (4.5)$$

The diffusion term is also evaluated with ∇z^h constant in $T_{i+1/2}$ and $\bar{\mu}_{i+1/2}$ the area weighted average μ.

$$\int_{T_{i+1/2}} \mu \nabla z^h \, da = A_{i+1/2}\, \bar{\mu}_{i+1/2} \nabla z^h_{i+1/2} \qquad (4.6)$$

This simplifies (4.0) considerably.

$$\frac{\partial}{\partial t}\int_{\Omega_j}\phi z^h\,da + \sum_{i=1}^{d(v_0)}\frac{1}{6}\vec{\mathbf{n}}_{i+1/2}\cdot(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)+\mathbf{F}(z_{i+1}^h)) = \sum_{i=1}^{d(v_0)}\frac{1}{2}\overline{\mu}_{i+1/2}\,\vec{\mathbf{n}}_{i+1/2}\cdot\nabla_{i+1/2}z^h \tag{4.7}$$

Equation (4.7) represents a Galerkin discretization of the model equation assuming piecewise linear functions. Note that as far as the geometry is concerned, only the exterior normals of Ω_j appear. Conspicuously absent are the normal vectors for interior edges. This strengthens our confidence that we can show an equivalence with a finite-volume discretization on *nonoverlapping* control volumes. To show this equivalence, note that the flux term can be manipulated using the identity $\sum_{i=1}^{d(v_0)}\vec{\mathbf{n}}_{i+1/2}=\mathbf{0}$ into a form in which the relevant geometry is any path connecting adjacent triangle centroids ($\mathbf{R}$ denotes the spatial position vector):

$$\begin{aligned}\sum_{i=1}^{d(v_0)}\frac{1}{6}\vec{\mathbf{n}}_{i+1/2}\cdot(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h)+\mathbf{F}(z_{i+1}^h)) &= \sum_{i=1}^{d(v_0)}\frac{1}{6}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot(\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2})\\ &= \sum_{i=1}^{d(v_0)}\frac{1}{6}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot\int_{\mathbf{R}_{i-1}}^{\mathbf{R}_{i+1}}\mathbf{n}\,dl\\ &= \sum_{i=1}^{d(v_0)}\frac{1}{2}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot\int_{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i-1})}^{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i+1})}\mathbf{n}\,dl\end{aligned} \tag{4.8}$$

The diffusion term also simplifies using this identity.

$$\sum_{i=1}^{d(v_0)}\frac{1}{2}\vec{\mathbf{n}}_{i+1/2}\cdot\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h = \sum_{i=1}^{d(v_0)}\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h\cdot\int_{\frac{1}{2}(\mathbf{R}_0+\mathbf{R}_i)}^{\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_{i+1})}\mathbf{n}\,dl \tag{4.9}$$

To obtain a single consistent path for the integrations appearing in equations(4.8) and (4.9) requires that the path pass through the centroid of each triangle and the mid-side of each interior edge. The path formed by connecting these points by line segments is precisely the median dual of the mesh. This dual completely covers the domain (no holes) and represents a consistent and conservative finite-volume discretization of the domain which is spatially equivalent to the Galerkin approximation. The scheme can now be written in a finite-volume form

$$\frac{\partial}{\partial t}\phi^h z^h\,da + \sum_{i=1}^{d(v_0)}(\mathbf{H}\cdot\vec{\mathbf{n}})_i = 0 \tag{4.10}$$

where $\mathbf{H}$ is the numerical flux of the finite-volume discretization

$$\begin{aligned}(\mathbf{H}\cdot\vec{\mathbf{n}})_i = &\frac{1}{2}(\mathbf{F}(z_0^h)+\mathbf{F}(z_i^h))\cdot\int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}}\mathbf{n}\,dl\\ &-\overline{\mu}_{i-1/2}\nabla_{i-1/2}z^h\cdot\int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}_i^m}\mathbf{n}\,dl\\ &-\overline{\mu}_{i+1/2}\nabla_{i+1/2}z^h\cdot\int_{\mathbf{R}_i^m}^{\mathbf{R}'_{i+1/2}}\mathbf{n}\,dl\end{aligned} \tag{4.11}$$

and $\mathbf{R}'_{i+1/2}$ is the centroid of $T_{i+1/2}$, $\mathbf{R}'_{i+1/2}=\frac{1}{3}(\mathbf{R}_0+\mathbf{R}_i+\mathbf{R}_{i+1})$ and $\mathbf{R}_i^m$ is the midpoint of the edge $e(v_0,v_i)$, $\mathbf{R}_i^m=\frac{1}{2}(\mathbf{R}_0+\mathbf{R}_i)$.

Conclusion: ***The spatial discretization produced by the Galerkin finite-element scheme with linear elements has an equivalent finite-volume discretization on nonoverlapping control volumes with bounding curves which pass through the centroid of triangles and midside of edges. One such set of control volumes satisfying these constraints is the median dual.***

We now need to ask if the time integrals produce identical "mass" matrices for the Galerkin finite-element and finite-volume schemes. The answer to this question is no. In fact, these matrices are not the same in one space dimension. The Galerkin mass matrix for a simple 1-D mesh with uniform spacing produces a row of the mass matrix with the following weights:

$$\frac{\partial}{\partial t}\int_{\Omega_j}\phi^h z^h dx = \frac{\partial}{\partial t}\Delta x\frac{1}{6}(z_{j-1}+4z_j+z_{j+1})$$

(Finite – Element)

The finite volume scheme on "median" dual produces the following weights:

$$\frac{\partial}{\partial t}\int_{\Omega_j} z^h dx = \frac{\partial}{\partial t}\Delta x\frac{1}{8}(z_{j-1}+6z_j+z_{j+1})$$

(Finite – Volume)

Although the finite-volume matrix gives better temporal stability, the finite-element mass matrix is more accurate.

4.3 Edge Formulas

The first term appearing on the right-hand-side of equation (4.10)

$$\frac{1}{2}(\mathbf{F}(z_0^h) + \mathbf{F}(z_i^h)) \cdot \int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}} \mathbf{n}\, dl$$

suggests a computer implementation using an *edge* data structure. The fluxes in this term are evaluated at the two endpoints of an edge. The geometrical terms could be evaluated edgewise if the midpoint of the edge and the centroids of the two adjacent cells are known. Recall that the edge data structure (described in section 1.3) for a 2-D mesh supplies this information for each interior edge of the mesh, i.e. the structure provides for each edge

(1) The two vertices which form the edge.

(2) The two adjacent cell centroids (or a pointer to centroid values) which share the edge.

More generally, if the solution is assumed to vary linearly within each triangle then edge formulas can be derived for discretized forms of the gradient, divergence, Hessian, and Laplacian operators. As we will see, the formulas can be derived from either a finite-volume or finite-element point-of-view with essentially identical results.

4.3a Gradient and Divergence Edge Formulas

As a first example, we will derive an edge formula for the integral averaged gradient of a function u, $\int \nabla u\, da$, for the the region Ω_0 described by the union of all triangles which share the vertex v_0, see figure 4.1a. If the discrete solution u^h varies linearly in each triangle T then the gradient is constant and the integration exact.

$$\int_{\Omega_0} \nabla u^h\, da = \sum_{T\in\Omega_0} (\nabla u^h)_T A_T \qquad (4.12)$$

Equation (4.12) would suggest computing the gradient in each triangle sharing v_0 and accumulating the area weighted sum. If integral averaged gradients are required at all vertices then the gradient in each triangle could be computed and the area weighted result scattered to the three vertices of the triangle for accumulation. We refer to this as the element-by-element approach. A Green's formula would suggest a different approach for the same task.

$$\int_{\Omega_0} \nabla u\, da = \oint_{\partial\Omega_0} u\,\mathbf{n}\, dl \qquad (4.13)$$

Identical results are obtained by approximating the right-hand-side of (4.13) by trapezoidal quadrature (exact for piecewise linear u^h)

$$\oint_{\partial\Omega_0} u^h\,\mathbf{n}\, dl = \sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_i^h + u_{i+1}^h)\vec{\mathbf{n}}_{i+1/2} \qquad (4.14)$$

where $\mathcal{I}_0 = \{1,2,...,6\}$ and $\vec{\mathbf{n}}_{i+1/2}$ is the vector perpendicular to the edge $e(v_i, v_{i+1})$ with magnitude equal to the length of the edge. The summation can be rearranged to yield

$$\oint_{\partial\Omega_0} u^h\,\mathbf{n}\, dl = \sum_{i\in\mathcal{I}_0} \frac{1}{2}u_i^h(\vec{\mathbf{n}}_{i+1/2}+\vec{\mathbf{n}}_{i-1/2}). \qquad (4.15)$$

A constant solution can be added to (4.15) since the gradient of a constant function is exactly zero in this discretization. In particular, we add the value of u^h at vertex v_0.

$$\oint_{\partial\Omega_0} u^h\,\mathbf{n}\, dl = \sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_0^h + u_i^h)(\vec{\mathbf{n}}_{i+1/2} + \vec{\mathbf{n}}_{i-1/2}) \qquad (4.16)$$

Once again using the fact that for any closed curve $\oint \mathbf{n}\, dl = \oint d\vec{\mathbf{n}} = 0$ which implies that

$$\vec{\mathbf{n}}_{i+1/2} + \vec{\mathbf{n}}_{i-1/2} = \int_{v_{i-1}}^{v_{i+1}} d\vec{\mathbf{n}}$$

for *any* path connecting v_{i-1} and v_{i+1}. This path integral represents a vector which is parallel in direction and three times the magnitude of the vector $\vec{\mathbf{n}}$ obtained by computing the integral for any simple path connecting the centroids of the two triangles which share the edge $e(v_0, v_i)$. $\int_{v_{i-1}}^{v_{i+1}} d\vec{\mathbf{n}} = 3\int_{v_{i'-1}}^{v_{i'}} d\vec{\mathbf{n}} = 3\vec{\mathbf{n}}_{0i}$. This reduces the gradient formula to the following form:

$$\oint_{\partial\Omega_0} u^h\,\mathbf{n}\, dl = \sum_{i\in\mathcal{I}_0} \frac{3}{2}(u_0^h + u_i^h)\vec{\mathbf{n}}_{0i} \qquad (4.17)$$

The vertex lumped average gradient at vertex is then given by

$$(\nabla u^h)_{v_0} = \frac{3}{A_{\Omega_0}} \sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_0^h + u_i^h)\vec{\mathbf{n}}_{0i}. \qquad (4.18)$$

It is well known that the region bounded by the "median" dual at vertex v_0, (shown in figure 4.1a) has an area A_0 which is exactly $\frac{1}{3}A_{\Omega_0}$. Therefore, using the median dual we obtain a formula which appears to represent some approximate quadrature of the right-hand-side of (4.13) on nonoverlapping regions.

$$(\nabla u^h)_{v_0} = \frac{1}{A_0} \sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_0^h + u_i^h)\vec{\mathbf{n}}_{0i} \qquad (4.19)$$

A naive interpretation of equation (4.19) would probably conclude that this equation is a rather poor approximation to (4.13). It is not obvious from (4.19) that the gradient of a linear function u is computed exactly. From the origin of this formula, we now know that this formula can be obtained from a trapezoidal quadrature on a slightly larger region and is exact within the class of linear polynomials.

Keep in mind that a constant solution could have been subtracted instead of added from equation (4.15) which would have given a different but equivalent form of (4.19).

$$(\nabla u^h)_{v_0} = \frac{1}{A_0}\sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_i^h - u_0^h)\vec{\mathbf{n}}_{0i} \qquad (4.20)$$

This formula does not appear to resemble any approximate quadrature of (4.13).

Equation (4.19) suggests an algorithm using an edge data structure which is quite different from the element-by-element method (4.12). The edge-based calculation consists of the following steps:

Sample Gradient Computation

(1) (Precomputation) For each edge $e(v_i, v_j)$ *gather* the centroid coordinates of the two adjacent cells, v_i' and v_j'.

(2) (Precomputation) For each edge *compute* the dual edge normal $\vec{\mathbf{n}}_{ij}$ from the centroid coordinates $\vec{\mathbf{n}}_{ij} = \int_{v_i'}^{v_j'} d\vec{\mathbf{n}}$. (Orient from v_i to v_j if $i < j$).

(3) For each edge $e(v_i, v_j)$ *gather* the values of the function at the two vertices, u_i^h and u_j^h.

(4) For each edge *compute* the arithmetic average and multiply by the dual edge normal, $\frac{1}{2}(u_i + u_j)\mathbf{n}_{ij}$.

(5) For each edge *scatter and accumulate* the result at vertex v_i.

(6) For each edge *negate, scatter and accumulate* the same result at vertex v_j.

(7) For each vertex *compute* the final gradient by dividing the accumulated result by area of the median dual, A_0.

This algorithm conforms perfectly within the edge data structure. In practice all the geometrical factors could be precomputed and stored in memory by edge thereby eliminating a gather. The sample algorithm described above serves as a template for all the remaining algorithms described in the rest of this section.

The gradient and divergence operators are related so that it is not surprising that the discretization of the divergence operator produces a similar formula:

$$\int_{\Omega_0} div(\mathbf{F}^h)da = \sum_{i\in\mathcal{I}_0}\frac{3}{2}(\mathbf{F}_0^h + \mathbf{F}_i^h)\cdot\vec{\mathbf{n}}_{0i} \qquad (4.21)$$

The Galerkin weighted finite element integrals with linear elements produce essentially identical results. In this case a piecewise linear weighting function ϕ^h is introduced (see figure 4.1b). The gradient and divergence formulas (introduced earlier)

$$\int_{\Omega_0} \phi^h\, \nabla u^h da = \sum_{i\in\mathcal{I}_0}\frac{1}{2}(u_0^h + u_i^h)\vec{\mathbf{n}}_{0i} \qquad (4.22)$$

$$\int_{\Omega_0} \phi^h\, div(\mathbf{F}^h)da = \sum_{i\in\mathcal{I}_0}\frac{1}{2}(\mathbf{F}_0^h + \mathbf{F}_i^h)\cdot\vec{\mathbf{n}}_{0i} \qquad (4.23)$$

differ from the previous formulas by a constant factor of 1/3. For example, if a lumped approximation to the left-hand-side of (4.22) is assumed, then (4.19) is recovered since

$$\int_{\Omega_0} \phi^h \nabla u^h da \approx (\nabla u^h)_{v_0} A_0. \qquad (4.24)$$

4.3b 2-D Hessian and Laplacian Edge Formulas

We begin by approximating the following matrix of second derivatives

$$\nabla\mu(\nabla u)^T = \begin{bmatrix} (\mu u_x)_x & (\mu u_y)_x \\ (\mu u_x)_y & (\mu u_y)_y \end{bmatrix} \qquad (4.25)$$

using a standard Galerkin approximation for the region Ω_0 formed from the union of all triangles that share the vertex v_0. To do so, multiply (4.25) by the weight function ϕ and perform integration by parts over Ω_0 assuming $\phi = 0$ on $\partial\Omega_0$.

$$\begin{aligned}\int_{\Omega_0} \phi\nabla\, \mu(\nabla u)^T da &= -\int_{\Omega_0} \mu(\nabla\phi)\,(\nabla u)^T da \\ &= -\sum_{i\in\mathcal{I}_0}\int_{T_{i+1/2}} \mu(\nabla\phi)\,(\nabla u)^T da\end{aligned} \qquad (4.26)$$

where $T_{i+1/2} \equiv simplex(v_0, v_i, v_{i+1})$. Using the notation of figure 4.2, gradients of the piecewise linear functions ϕ^h (figure 4.1b) and u^h are

$$(\nabla\phi^h)_{T_{i+1/2}} = -\frac{1}{2A_{i+1/2}}\vec{\mathbf{n}}_{i+1/2} \qquad (4.27)$$

and

$$(\nabla u^h)_{T_{i+1/2}} = \frac{-1}{2A_{i+1/2}}(u_0^h \vec{\mathbf{n}}_{i+1/2} + u_i^h \vec{\mathbf{n}}_{i+1} - u_{i+1}^h \vec{\mathbf{n}}_i) \tag{4.28}$$

where $A_{i+1/2}$ is the area of $T_{i+1/2}$ and $\vec{\mathbf{n}}_{i+1/2}$ is the vector normal to the edge $e(v_i, v_{i+1})$ with magnitude equal to the length of the edge.

For piecewise linear u^h, the gradient is constant in each triangle. The integral average matrix of second derivatives simplifies to the following form:

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla \mu (\nabla u^h)^T da &= \sum_{i \in \mathcal{I}_0} \frac{1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} \int_{T_{i+1/2}} \mu (\nabla u^h)^T da \\ &= \sum_{i \in \mathcal{I}_0} \frac{1}{2A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (\nabla u^h)^T_{T_{i+1/2}} \int_{T_{i+1/2}} \mu da \\ &= \sum_{i \in \mathcal{I}_0} \frac{\overline{\mu}_{i+1/2}}{2} \vec{\mathbf{n}}_{i+1/2} (\nabla u^h)^T_{T_{i+1/2}}\end{aligned} \tag{4.29}$$

where $\overline{\mu}_{i+1/2}$ is the integral average of μ in $T_{i+1/2}$ Inserting the triangle gradient formula, we obtain a discretized formula for the Galerkin integral.

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla \mu (\nabla u^h)^T da &= -\frac{1}{4} \sum_{i \in \mathcal{I}_0} \frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (u_0^h \vec{\mathbf{n}}_{i+1/2}^T \\ &\quad + u_i^h \vec{\mathbf{n}}_{i+1}^T - u_{i+1}^h \vec{\mathbf{n}}_i^T)\end{aligned} \tag{4.30}$$

Regrouping terms and removal of a constant solution yields the following simplified form

$$\begin{aligned}\int_{\Omega_0} \phi^h \nabla \mu (\nabla u^h)^T da &= \int_{\Omega_0} \nabla \mu (\nabla (u^h - u_0^h))^T da \\ &= \sum_{i \in \mathcal{I}_0} M_i (u_i^h - u_0^h)\end{aligned} \tag{4.31}$$

with

$$M_i = -\frac{1}{4} \left[\frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}} \vec{\mathbf{n}}_{i+1/2} (\vec{\mathbf{n}}_{i+1})^T - \frac{\overline{\mu}_{i-1/2}}{A_{i-1/2}} \vec{\mathbf{n}}_{i-1/2} (\vec{\mathbf{n}}_{i-1})^T \right]. \tag{4.32}$$

Even though this formula is very simple, it is not compatible with the edge data structure mentioned earlier. Using some simple identities, we will now rewrite the weight formula in a form which is compatible with the edge data structure.

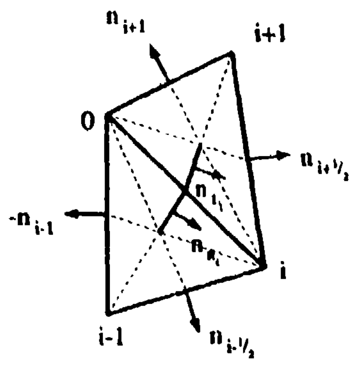


Figure 4.3 Local geometry configuration.

Referring to figure 4.3, we have the following vector identities:

$$\vec{\mathbf{n}}_{i-1/2} = 3\vec{\mathbf{n}}_{R_i} - \frac{1}{2}\vec{\mathbf{n}}_i, \quad \vec{\mathbf{n}}_{i-1} = 3\vec{\mathbf{n}}_{R_i} + \frac{1}{2}\vec{\mathbf{n}}_i \tag{4.33}$$

and similarly

$$\vec{\mathbf{n}}_{i+1/2} = 3\vec{\mathbf{n}}_{L_i} + \frac{1}{2}\vec{\mathbf{n}}_i, \quad \vec{\mathbf{n}}_{i+1} = -3\vec{\mathbf{n}}_{L_i} + \frac{1}{2}\vec{\mathbf{n}}_i. \tag{4.34}$$

It is useful to decompose the tensor product terms into symmetric and skew-symmetric parts, for example:

$$\begin{aligned}-\vec{\mathbf{n}}_{i-1/2}(\vec{\mathbf{n}}_{i-1})^T &= \underbrace{(\frac{1}{4}\vec{\mathbf{n}}_i \vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_R \vec{\mathbf{n}}_R^T)}_{Symmetric} \\ &\quad + \underbrace{\frac{3}{2}(\vec{\mathbf{n}}_i \vec{\mathbf{n}}_R^T - \vec{\mathbf{n}}_R \vec{\mathbf{n}}_i^T)}_{Skew-symmetric}\end{aligned}$$

and

$$\begin{aligned}\vec{\mathbf{n}}_{i+1/2}(\vec{\mathbf{n}}_{i+1})^T &= \underbrace{(\frac{1}{4}\vec{\mathbf{n}}_i \vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_L \vec{\mathbf{n}}_L^T)}_{Symmetric} \\ &\quad + \underbrace{\frac{3}{2}(\vec{\mathbf{n}}_i \vec{\mathbf{n}}_L^T - \vec{\mathbf{n}}_L \vec{\mathbf{n}}_i^T)}_{Skew-symmetric}\end{aligned}$$

Upon dividing by the area terms, some simple algebra reveals that

$$\frac{\vec{\mathbf{n}}_{i-1/2}(\vec{\mathbf{n}}_{i-1})^T}{A_{i-1/2}} = \frac{(9\vec{\mathbf{n}}_{R_i}\vec{\mathbf{n}}_{R_i}^T - \frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T)}{A_{i-1/2}} + \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{4.35}$$

and similarly

$$\frac{\vec{\mathbf{n}}_{i+1/2}(\vec{\mathbf{n}}_{i+1})^T}{A_{i+1/2}} = \frac{(\frac{1}{4}\vec{\mathbf{n}}_i\vec{\mathbf{n}}_i^T - 9\vec{\mathbf{n}}_{L_i}\vec{\mathbf{n}}_{L_i}^T)}{A_{i+1/2}} + \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}. \tag{4.36}$$

So in summary we have that

$$\begin{aligned} M_i = & -\frac{1}{4}\left[\frac{\overline{\mu}_{i+1/2}}{A_{i+1/2}}\vec{n}_{i+1/2}(\vec{n}_{i+1})^T \right. \\ & \left. - \frac{\overline{\mu}_{i-1/2}}{A_{i-1/2}}\vec{n}_{i-1/2}(\vec{n}_{i-1})^T\right] \\ = & -\frac{1}{4}\left[\overline{\mu}_{i+1/2}\frac{\frac{1}{4}\vec{n}_i\vec{n}_i^T - 9\vec{n}_{L_i}\vec{n}_{L_i}^T}{A_{i+1/2}} \right. \\ & +\overline{\mu}_{i-1/2}\frac{\frac{1}{4}\vec{n}_i\vec{n}_i^T - 9\vec{n}_{R_i}\vec{n}_{R_i}^T}{A_{i-1/2}} \\ & \left. +(\overline{\mu}_{i+1/2} - \overline{\mu}_{i-1/2})\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}\right] \end{aligned} \quad (4.37)$$

The second form is compatible with an edge data structure where edge vertices and adjacent cell centroids are known.

4.3c 2-D Edge Discretization of $\nabla \cdot \mu\nabla u$

Calculation of this term amounts to summing diagonal entries of the previous result. Cancellation of terms leaves a reduced form.

$$\int_{\Omega_0} \phi^h \nabla \cdot \mu\nabla u^h \, da = \text{Trace} \int_{\Omega_0} \phi^h \nabla \, \mu(\nabla u^h)^T \, da = \sum_{i\in\mathcal{I}_0} W_i(u_i^h - u_0^h) \quad (4.38)$$

$$W_i = -\frac{1}{4}\left[\overline{\mu}_{i+1/2}\frac{\vec{n}_{i+1/2}\cdot\vec{n}_{i+1}}{A_{i+1/2}} + \overline{\mu}_{i-1/2}\frac{\vec{n}_{i-1/2}\cdot\vec{n}_{i-1}}{A_{i-1/2}}\right] \quad (4.39)$$

The area of a triangle can be expressed in terms of the magnitude of the cross product of the scaled edge normals.

$$A_{i+1/2} = \frac{1}{2}|\vec{n}_{i+1/2} \times \vec{n}_{i+1}|$$

$$A_{i-1/2} = \frac{1}{2}|\vec{n}_{i-1/2} \times \vec{n}_{i-1}|$$

$$W_i = -\frac{1}{2}\left[\overline{\mu}_{i+1/2}\frac{(\vec{n}_{i+1/2}\cdot\vec{n}_{i+1})}{|\vec{n}_{i+1/2}\times\vec{n}_{i+1}|} - \overline{\mu}_{i-1/2}\frac{(\vec{n}_{i-1/2}\cdot\vec{n}_{i-1})}{|\vec{n}_{i-1/2}\times\vec{n}_{i-1}|}\right] \quad (4.40)$$

Finally we can express the dot and cross products in terms of the local angles as sketched in figure 4.4.

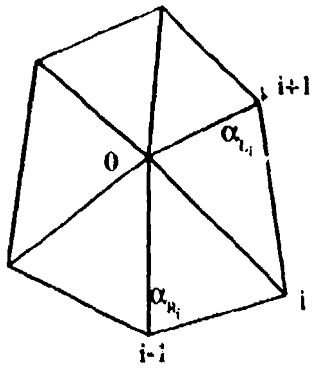


Figure 4.4 Local angles for triangles sharing edge $e(v_0, v_i)$.

$$\frac{\vec{n}_{i+1/2}\cdot\vec{n}_{i+1}}{|\vec{n}_{i+1/2}\times\vec{n}_{i+1}|} = -\frac{\cos(\alpha_{L_i})}{\sin(\alpha_{L_i})} = -\text{cotan}(\alpha_{L_i}) \quad (4.41)$$

and

$$-\frac{\vec{n}_{i-1/2}\cdot\vec{n}_{i-1}}{|\vec{n}_{i-1/2}\times\vec{n}_{i-1}|} = -\frac{\cos(\alpha_{R_i})}{\sin(\alpha_{R_i})} = -\text{cotan}(\alpha_{R_i}) \quad (4.42)$$

Inserting these formulas yields a particularly simple form of the weight factors W_i:

$$W_i = \frac{1}{2}\left[\overline{\mu}_{i+1/2}\text{cotan}(\alpha_{L_i}) + \overline{\mu}_{i-1/2}\text{cotan}(\alpha_{R_i})\right] \quad (4.43)$$

Equation (4.43) is particularly useful in theoretical studies.

4.3d 3-D Hessian and Laplacian Edge Formulas

As in the 2-D case, we begin with the Galerkin integral equation for the Hessian-like matrix of derivatives.

$$\int_\Gamma \phi^h \nabla \, \mu(\nabla u^h)^T \, dv = -\int_\Gamma \mu(\nabla\phi^h)\,(\nabla u^h)^T \, dv$$

In this formula is the volume formed by the union of all tetraheara that share vertex v_0. Following a procedure identical to the 2-D case, we can derive the analogous 3-D edge formula for the matrix of second derivatives

$$\int_{V_{\Gamma_0}} \phi^h \nabla \, \mu(\nabla u^h)^T \, dv = \sum_{i\in\mathcal{I}_0} M_i(u_i - u_0) \quad (4.44)$$

where

$$M_i = -\frac{1}{9}\sum_{k=1}^{d(v_0,v_i)} \frac{\overline{\mu}_{k+1/2}}{V_{k+1/2}}\vec{s}_{k+1/2}(\vec{s}_{k+1/2})^T \quad (4.45)$$

$\mathcal{I}_0$ is the set of indices of all adjacent neighbors of v_0 connected by incident edges, k a local cyclic index describing the associated vertices which form a polygon of degree $d(v_0, v_i)$ surrounding the edge $e(v_0, v_i)$. The subscript $k + 1/2$ indicates quantities associated with the tetrahedron with vertices v_0, v_i, v_k and v_{k+1} as shown in figure 4.5.

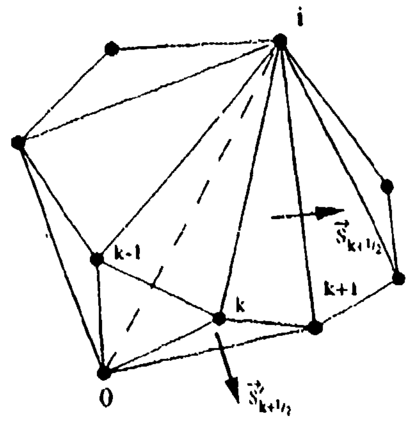


Figure 4.5 Set of tetrahedra sharing edge $e(v_0, v_i)$ with local cyclic index k.

4.3e 3-D Edge Discretization of $\nabla \cdot \mu \nabla u$

Following the same procedure as in 2-D, we obtain:

$$\int_{\Gamma_0} \phi^h \nabla \cdot \mu \nabla u^h \; dv = \mathrm{Trace} \int_{\Gamma_0} \phi^h \nabla \mu (\nabla u^h)^T \, dv$$
$$= \sum_{i \in \mathcal{I}_0} W_i (u_i - u_0) \tag{4.46}$$

where

$$W_i = -\frac{1}{9} \sum_{k=1}^{d(v_0, v_i)} \frac{\overline{\mu}_{k+1/2}}{V_{k+1/2}} \vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2} \tag{4.47}$$

It can be shown that the volume of a tetrahedron is given by

$$V_{k+1/2} = \frac{2}{3} \frac{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|}{|\Delta \mathbf{R}_{k+1/2}|} \tag{4.48}$$

where $|\Delta \mathbf{R}_{k+1/2}|$ is the length of the edge shared by the faces associated with $\vec{s}_{k+1/2}$ and $\vec{s}'_{k+1/2}$.

$$W_i = -\frac{1}{6} \sum_{k=1}^{d(v_0, v_i)} \overline{\mu}_{k+1/2} |\Delta R_{k+1/2}| \frac{\vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2}}{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|} \tag{4.49}$$

Finally we can rewrite the dot and cross product in terms of the cotangent of the face angle.

$$\frac{\vec{s}_{k+1/2} \cdot \vec{s}'_{k+1/2}}{|\vec{s}_{k+1/2} \times \vec{s}'_{k+1/2}|} = -\frac{\cos(\alpha_{k+1/2})}{\sin(\alpha_{k+1/2})}$$
$$= -\operatorname{cotan}(\alpha_{k+1/2})$$

As in the 2-D case, the weights W_i now have a particularly simple form:

$$W_i = \frac{1}{6} \sum_{k=1}^{d(v_0, v_i)} \overline{\mu}_{k+1/2} |\Delta R_{k+1/2}| \operatorname{cotan}(\alpha_{k+1/2}) \tag{4.50}$$

4.4 Godunov Finite-Volume Schemes

In this section, we consider upwind algorithms for scalar hyperbolic equations. In particular, we concentrate on upwind schemes based on Godunov's method [34] and defer the discussion of "upwind" schemes based on the fluctuation decomposition method or the Petrov-Galerkin formulation (SUPG, GLS) to the lectures of Profs. Deconinck, Hughes, and Johnson.

The development presented here follows many of the ideas developed previously for structured meshes. For example, in the extension of Godunov's scheme to second order accuracy in one space dimension, van Leer [35] developed an advection scheme based on the [illegible] of discontinuous piecewise linear [illegible] together with Lagrangian hydrodynamics [illegible]. Soon thereafter, Colella and Woodward [illegible] and Woodward and Colella [37] further extended these ideas to include discontinuous piecewise parabolic approximations with Eulerian or Lagrangian hydrodynamics. Harten et. al. [38,39] later extended related schemes to arbitrary order and clarified the entire process. These techniques have been applied to structured meshes in multiple space dimensions by applying one-dimensional-like schemes along individual coordinate lines. This has proven to be a highly successful approximation but does not directly extend to unstructured meshes. In reference [40], we proposed a scheme for multi-dimensional reconstruction on unstructured meshes using discontinuous piecewise linear distributions of the solution in each control volume. Monotonicity of the reconstruction was enforced using a limiting procedure similar to that proposed by van Leer [35] for structured grids. In a later paper (Barth and Frederickson [41]), we developed numerical schemes for unstructured meshes utilizing a reconstruction algorithm of arbitrary order. Portions of the discussion presented here is taken from these papers.

4.4a Generalized Godunov Scheme

We begin by considering the integral conservation law for some domain, Ω and its tessellation $\mathcal{T}(\Omega)$ comprised of cells, c_j, $\Omega = \cup c_j$, $c_k \cap c_j = \emptyset$, $k \neq j$. The integral equation is valid for the entire domain Ω as well as in each cell (or possibly dual cell):

$$\frac{\partial}{\partial t}\int_{c_j} u\, da + \int_{\partial c_j} \mathbf{F}(u)\cdot \mathbf{n}\, dl = 0 \qquad (4.51a)$$

Fundamental to Godunov's method is the cell average of the solution, $\overline{u}$, in each cell.

$$\int_{c_j} u\ da = \overline{u}_j A_j \qquad (4.52)$$

In Godunov's method and the higher order accurate extension considered here, these cell averages are treated as the fundamental unknowns (degrees of freedom).

$$\frac{\partial}{\partial t}(\overline{u}_j A_j) + \int_{\partial c_j} \mathbf{F}(u)\cdot \mathbf{n}\, dl = 0 \qquad (4.51b)$$

The solution algorithm for (4.51b) is a relatively standard procedure for extensions of Godunov's scheme in Eulerian coordinates [34-39]. The basic idea is to start with piecewise constant data in each cell with value equal to the integral cell average. Using information from cell averages, $k-th$ order piecewise polynomials are *reconstructed*:

$$u^k(x,y) = \sum_{m+n\leq k} \alpha_{(m,n)} P_{(m,n)}(x-x_c, y-y_c) \qquad (4.53)$$

where $P_{(m,n)}(x-x_c, y-y_c) = (x-x_c)^m (y-y_c)^n$ and (x_c, y_c) is the cell centroid. The process of reconstruction amounts to finding the polynomial coefficients, $\alpha_{(m,n)}$. Near steep gradients and discontinuities, these polynomial coefficients maybe altered based on monotonicity arguments. Because the reconstructed polynomials vary discontinuously from cell to cell, a unique value of the solution does not exist at cell interfaces. This nonuniqueness is resolved via exact or approximate solutions of the Riemann problem. In practice, this is accomplished by supplanting the true flux function in (4.51) with a numerical flux function (described below) which produces a single unique flux given two solution states. Once the flux integral in (4.51) is carried out (either exactly or by numerical quadrature), the cell average of the solution can be evolved in time. In most cases, standard techniques for integrating ODE equations are used for the time evolution, i.e. Euler implicit, Euler explicit, Runge-Kutta. The result of the evolution process is a new collection of cell averages. The process can then be repeated. The process can be summarized in the following steps:

(1) **Reconstruction in Each Cell**: Given integral cell averages in all cells, reconstruct piecewise polynomial coefficients $\alpha_{(m,n)}$ for use in equation (4.51). For solutions containing discontinuities and/or steep gradients, monotonicity enforcement may be required.

(2) **Flux Evaluation on Each Edge**: Consider each cell boundary, ∂c_j, to be a collection of edges (or dual edges) from the mesh. Along each edge (or dual edge), perform a high order accurate flux quadrature.

(3) **Evolution in Each Cell**: Collect flux contributions in each cell and evolve in time using any time stepping scheme, i.e., Euler explicit, Euler implicit, Runge-Kutta, etc. The result of this process is once again cell averages.

By far, the most difficult of these steps is the polynomial reconstruction given cell averages. In the following paragraphs, we describe design criteria for a general reconstruction operator.

Reconstruction

The reconstruction operator serves as a finite-dimensional (possibly pseudo) inverse of the cell-averaging operator $\mathbf{A}$ whose j-th component $\mathbf{A}_j$ computes the cell average of the solution in c_j.

$$\overline{u}_j = \mathbf{A}_j u = \frac{1}{a_j}\int_{c_j} u(x,y)\, da \qquad (4.54)$$

In addition, we place the following additional requirements:

(1) **Conservation of the mean**: Simply stated, given cell averages $\overline{u}$, we require that all polynomial reconstructions u^k have the correct cell average.

$$\text{if } \ u^k = \mathbf{R}^k\overline{u} \ \text{ then } \ \overline{u} = \mathbf{A}u^k$$

This means that $\mathbf{R}^k$ is a right inverse of the averaging operator $\mathbf{A}$.

$$\mathbf{A}\mathbf{R}^k = I \qquad (4.55)$$

Conservation of the mean has an important implication. Unlike finite-element schemes, *Godunov schemes have a diagonal mass matrix.*

(2) **k-exactness:** We say that a reconstruction operator $\mathbf{R}^k$ is *k-exact* if $\mathbf{R}^k\mathbf{A}$ reconstructs polynomials of degree k or less exactly.

$$\text{if} \quad u \in \mathcal{P}_k \quad \text{and} \quad \overline{u} = \mathbf{A}u, \quad \text{then} \quad u^k = \mathbf{R}^k\overline{u} = u$$

In other words, $\mathbf{R}^k$ is a left-inverse of $\mathbf{A}$ restricted to the space of polynomials of degree at most k.

$$\mathbf{R}^k\mathbf{A}\bigg|_{\mathcal{P}_k} = I \qquad (4.56)$$

This insures that exact solutions contained in $\mathcal{P}_k$ are in fact solutions of the discrete equations. For sufficiently smooth solutions, the property of k-exactness also issures that when piecewise polynomials are evaluated at cell boundaries, the difference between solution states diminishes with increasing k at a rate proportional to h^{k+1} were h is a maximum diameter of the two cells. Figure 4.6a shows a global quartic polynomial $u \in \mathcal{P}_4$ which has been averaged in each interval.

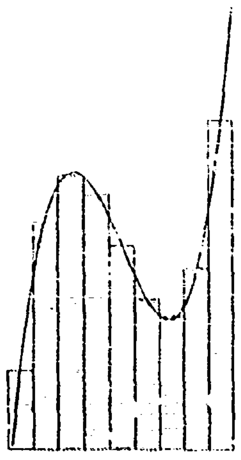


Figure 4.6a Cell averaging of quartic polynomial.

Figure 4.6b shows a quadratic reconstruction $u^k \in \mathcal{P}_2$ given the cell averages. Close inspection of figure 4.6b reveals small jumps in the piecewise polynomials at interval boundaries. These jumps would decrease even more for cubics and vanish altogether for quartic reconstruction. Property (1) requires that the area under each piecewise polynomial is exactly equal to the cell average.

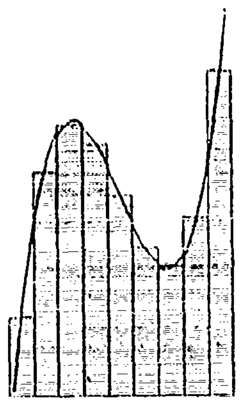


Figure 4.6b Piecewise quadratic reconstruction.

Flux Evaluation

The task here is to evaluate the flux integral appearing in (4.51).

$$\int_{\partial c_j} \mathbf{F}(u) \cdot \mathbf{n}\, dl \qquad (4.57)$$

At cell interfaces, two distinct values of the solution can be obtained anywhere on the boundary of the control volume by direct evaluation of the piecewise polynomials in the two cells sharing the interface. For brevity, the states will be denoted by u^+ and u^- which should be interpreted as $(u^k)^+$ and $(u^k)^-$ where $\pm$ refers to which piecewise polynomial was used in the evaluation. Rather than use a numerical flux function derived from the exact solution of the Riemann problem, we prefer numerical flux functions based on mean value linearizations. As we will see, this actually makes certain stability proofs much clearer. Define $f(u, \mathbf{n}) = \mathbf{F}(u) \cdot \mathbf{n}$ and $a(u, \mathbf{n}) = f(u, \mathbf{n})'$, the mean value flux function is given by

$$\begin{aligned} \mathbf{h}(u^+, u^-, \mathbf{n}) = & \frac{1}{2}\left(f(u^+, \mathbf{n}) + f(u^-, \mathbf{n})\right) \\ & - \frac{1}{2}|a(\tilde{u}, \mathbf{n})|\left(u^+ - u^-\right) \end{aligned} \qquad (4.58)$$

where $f(u^+) - f(u^-) = a(\tilde{u}, \mathbf{n})(u^+ - u^-)$ and $\tilde{u} = \theta u^- + (1-\theta)u^+$ for some $\theta \in [0, 1]$. Using the numerical flux function, we approximate (4.57) by

$$\int_{\partial c_j} \mathbf{h}(u^+, u^-, \mathbf{n})\, dl$$

In practice, this flux integral is never evaluated exactly, except when the data is piecewise constant. When piecewise linear functions are used, a midpoint quadrature formula is usually employed. This is used rather than the slightly more accurate trapezoidal quadrature because it requires only one flux evaluation per edge segment while the trapezoidal quadrature requires two. When considering schemes with reconstruction order k greater than one, we suggest in [41] that Gauss quadrature formulas be used. Recall that N point Gauss quadrature formulas integrate $2N-1$ polynomials exactly. These quadrature formulas give the highest accuracy for the lowest number of function evaluations. For the k-exact reconstruction discussed below, $N \geq (k+1)/2$ point Gauss quadrature formulas are used.

4.5 k-exact Reconstruction

In this section, a brief account is given of the reconstruction scheme presented in Barth and Frederickson [41] for arbitrary order reconstruction. Upon first inspection, the use of high order reconstruction appears to be an expensive proposition. The present reconstruction strategy optimizes the efficiency of the reconstruction by precomputing as a *one time* preprocessing step the set of weights $\mathbf{W}_j$ in each cell $\mathbf{c}_j$ with neighbor set $\mathcal{N}_{c_j}$ such that

$$\alpha_{(m,n)} = \sum_{i \in \mathcal{N}_j} W_{(m,n)i} \overline{u}_i \qquad (4.59)$$

where $\alpha_{(m,n)}$ are the polynomial coefficients. This effectively reduces the problem of reconstruction to multiplication of predetermined weights and cell averages to obtain polynomial coefficients.

During the preprocessing to obtain the reconstruction weights $\mathbf{W}_j$ a coordinate system with origin at the centroid of $\mathbf{c}_j$ is assumed to minimize roundoff errors. To insure that the reconstruction is invariant to affine transformations, we then temporarily transform (rotate and scale) to another coordinate system $(\overline{x}, \overline{y})$ which is normalized to the cell $\mathbf{c}_j$

$$\begin{bmatrix} \overline{x} \\ \overline{y} \end{bmatrix} = \begin{bmatrix} \mathbf{D}_{1,1} & \mathbf{D}_{1,2} \\ \mathbf{D}_{2,1} & \mathbf{D}_{2,2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

with the matrix $\mathbf{D}$ is chosen so that

$$\mathbf{A}_j(\overline{x}^2) = \mathbf{A}_j(\overline{y}^2) = 1$$

$$\mathbf{A}_j(\overline{x}\,\overline{y}) = \mathbf{A}_j(\overline{y}\,\overline{x}) = 0$$

Polynomials on $\mathbf{c}_j$ are temporarily represented using the polynomial basis functions

$$\overline{P} = [1, \overline{x}, \overline{y}, \overline{x}^2, \overline{x}\overline{y}, \overline{y}^2, \overline{x}^3, \ldots].$$

Note that polynomials in this system are easily transformed to the standard cell-centroid basis

$$\overline{x}^m \overline{y}^n =$$

$$\sum_{s+t \leq k} \binom{m}{s} \binom{n}{t} \mathbf{D}_{1,1}^s \mathbf{D}_{1,2}^{m-s} \mathbf{D}_{2,1}^t \mathbf{D}_{2,2}^{n-t} x^{s+t} y^{m+n-s-t}$$

Since $0 \leq s+t \leq k$ and $0 \leq m+n-s-t \leq k$, we can reorder and rewrite in terms of the standard and transformed basis polynomials

$$\overline{P}_{(m,n)} = \sum_{s+t \leq k} G_{m,n}^{s,t} P_{(s,t)} \qquad (4.60)$$

Satisfaction of conservation of the mean is guaranteed by introducing into the transformed coordinate system *zero mean* basis polynomials $\overline{P}^0$ in which all but the first have zero cell average, i.e $\overline{P}^0 = [1, \overline{x}, \overline{y}, \overline{x}^2 - 1, \overline{x}\,\overline{y}, \overline{y}^2 - 1, \overline{x}^3 - A_j(\overline{x}^3), \ldots]$. Note that using these polynomials requires a minor modification of (4.60) but retains the same form:

$$\overline{P}^0_{(m,n)} = \sum_{s+t \leq k} \overline{G}_{m,n}^{s,t} P_{(s,t)} \qquad (4.61)$$

Given this preparatory work, we are now ready to describe the formulation of the reconstruction algorithm.

Minimum Energy (Least-Squares) Reconstruction

We note that the set of cell neighbors $\mathcal{N}_j$ must contain at least $(k+1)(k+2)/2$ cells $\mathbf{c}_j$ if the reconstruction operator $\mathbf{R}_j^k$ is to be k-exact. That $(k+1)(k+2)/2$ cells is not sufficient in all situations is easily observed. If, for example, the cell-centers all lie on a single straight line one can find a linear function u such that $\mathbf{A}_j(u) = 0$ for every cell $\mathbf{c}_j$, which means that reconstruction of u is impossible. In other cases a k-exact reconstruction operator $\mathbf{R}_j^k$ may exist, but due to the geometry may be poorly conditioned.

Our approach is to work with a slightly larger support containing more than the minimum number of cells. In this case the operator $\mathbf{R}_j^k$ is likely to be nonunique, because various subsets would be able to support reconstruction operators of degree k. Although all would reproduce a polynomial of degree k exactly, if we disregard roundoff, they would differ in their treatment of nonpolynomials, or of polynomials of degree higher than k. Any k-exact reconstruction operator $\mathbf{R}_j^k$ is a weighted average of these basic ones. Our approach is to choose the one of minimum Frobenius norm. This operator is optimal, in a certain sense, when the function we are reconstructing is not exactly a polynomial of degree k, but one that has been perturbed by the addition of Gaussian noise, for it minimizes the expected deviation from the unperturbed polynomial in a certain rather natural norm.

As we begin the formulation of the reconstruction preprocessing algorithm, the reader is reminded that the task at hand is to calculate the weights $\mathbf{W}_j$ for each cell $\mathbf{c}_j$ which when applied via (4.59) produce piecewise polynomial approximations. We begin by first rewriting the piecewise polynomial (4.53) for cell $\mathbf{c}_j$ in terms of the reconstruction weights (4.59)

$$u^k(x,y) = \sum_{m+n\le k} P_{(m,n)} \sum_{i\in\mathcal{N}_{c_j}} W_{(m,n)i}\overline{u}_i$$

or equivalently

$$u^k(x,y) = \sum_{i\in\mathcal{N}_{c_j}} \overline{u}_i \sum_{m+n\le k} W_{(m,n)i}P_{(m,n)}$$

Polynomials of degree k or less are equivalently represented in the transformed coordinate system using zero mean polynomials

$$u^k(x,y) = \sum_{i\in\mathcal{N}_{c_j}} \overline{u}_i \sum_{m+n\le k} W'_{(m,n)i}\overline{P}^0_{(m,n)} \quad (4.62)$$

Using (4.61), we can relate weights in the transformed system to weights in the original system

$$W_{(s,t)i} = \sum_{m+n\le k} \overline{G}^{s,t}_{m,n} W'_{(m,n),i} \quad (4.63)$$

We satisfy k-exactness by requiring that (4.62) is satisfied for all linear combinations of $\overline{P}^0_{(s,t)}(x,y)$ such that $s+t\le k$. In particular, if $u^k(x,y) = \overline{P}^0_{(s,t)}(x,y)$ for some $s+t\le k$ then

$$\overline{P}^0_{(s,t)}(x,y) = \sum_{m+n\le k} \overline{P}^0_{(m,n)} \sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i}A_i(\overline{P}^0_{(s,t)})$$

This is satisfied if for all $s+t, m+n\le k$

$$\sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i}A_i(\overline{P}^0_{(s,t)}) = \delta^{st}_{mn}$$

Transforming basis polynomials back to the original coordinate system we have

$$\sum_{i\in\mathcal{N}_{c_j}} W'_{(m,n)i} \sum_{u+v\le k} \overline{G}^{u,v}_{s,t} A_i(P_{(u,v)}) = \delta^{st}_{mn} \quad (4.64)$$

This can be locally rewritten in matrix form as

$$\mathbf{W}'_j\mathbf{A}'_j = \mathbf{I} \quad (4.65a)$$

and transformed in terms of the standard basis weights via

$$\mathbf{W}_j = \mathbf{G}\mathbf{W}'_j \quad (4.65b)$$

Note that $\mathbf{W}'_j$ is a $(k+1)(k+2)/2$ by $\mathcal{N}_j$ matrix and $\mathbf{A}'_j$ has dimensions $\mathcal{N}_j$ by $(k+1)(k+2)/2$. To solve (4.65a) in the optimum sense described above, an $\mathbf{L}_j\mathbf{Q}_j$ decomposition of $\mathbf{A}'_j$ is performed where the orthogonal matrix $\mathbf{Q}_j$ and the lower triangular matrix $\mathbf{L}_j$ have been constructed using a modified Gram-Schmidt algorithm (or a sequence of Householder reflections). The weights $\mathbf{W}'_j$ are then given by

$$\mathbf{W}'_j = \mathbf{Q}^*_j\mathbf{L}^{-1}_j$$

Applying (4.63) these weights are transformed to the standard centroid basis and the preprocessing step is complete.

We now show a few results presented earlier in reference [41]. The first calculation involves the reconstruction of a sixth order polynomial with random normalized coefficients which has been cell averaged onto a random mesh. Figures 4.7a-b show a sample mesh and the absolute L_2 error of the reconstruction for various meshes and reconstruction degree.

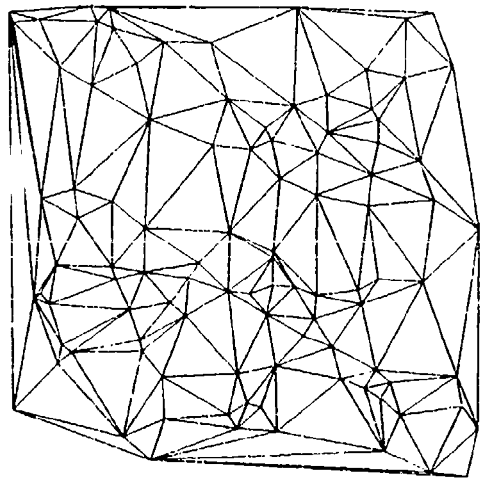


Figure 4.7a Random mesh.

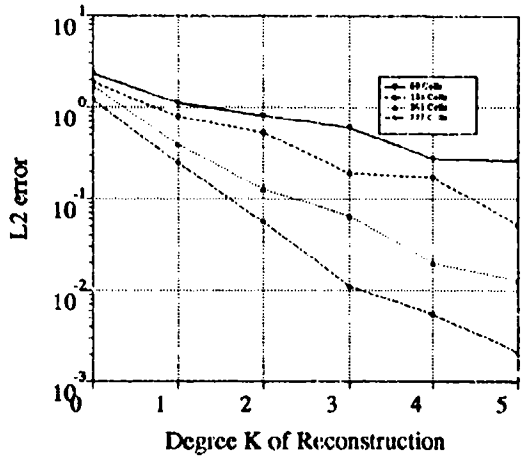


Figure 4.7b L_2 error of reconstruction.

The reconstruction algorithm has also been tested on more realistic problems. Figures 4.8a-c show a mesh and reconstructions (linear and quadratic) of a cell averaged density field corresponding to a Ringleb flow, an exact hodograph solution of the gasdynamic equations, see [42].

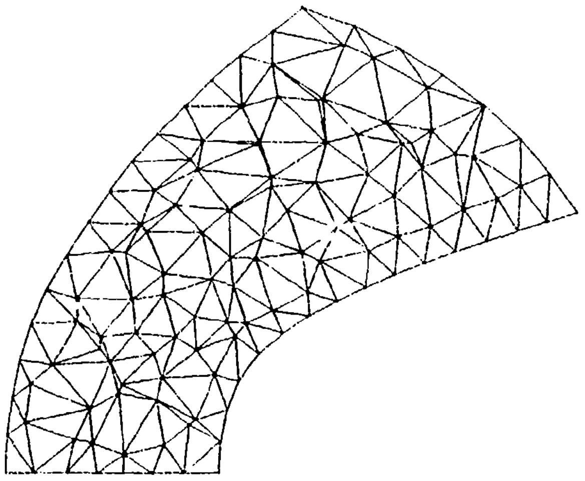


Figure 4.8a Randomized mesh for Ringleb flow.

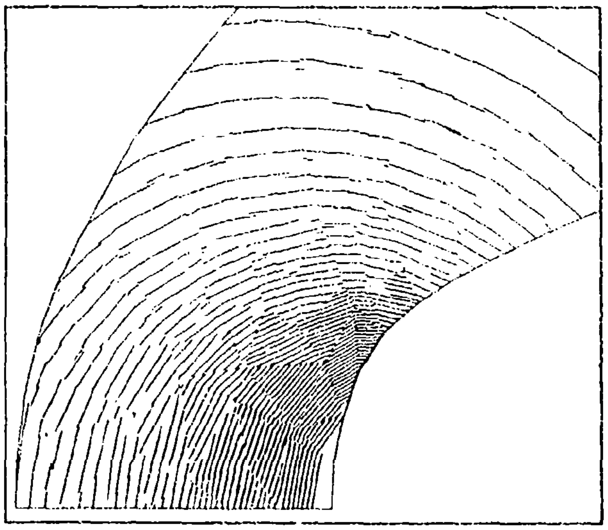


Figure 4.8b Piecewise linear reconstruction of Ringleb flow.

The reader should note that the use of piecewise contours gives a crude visual critique as to how well the solution is represented by the piecewise polynomials. The improvement from linear to quadratic is dramatic in the case of Ringleb flow. A later section will show actual numerical solutions computed using this reconstruction operator.

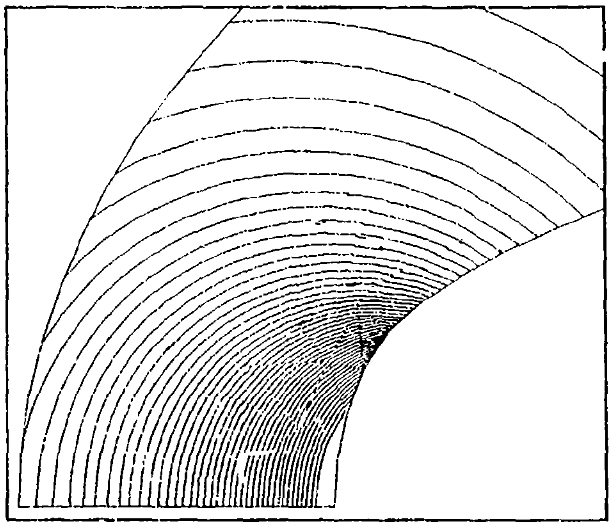


Figure 4.8c Piecewise quadratic reconstruction of Ringleb flow.

4.6 Upwind Advection Scheme with $k = 0$ Reconstruction

This is the simplest (first order) approximation in which the polynomial behavior in each cell, c_j, is a constant value equal to the cell average.

$$u^{k=0}(x,y) = \overline{u}_j \quad \text{for } u^k \in c_j \tag{4.66}$$

The flux formula then simplifies to the following form (for clarity $\overline{u}_j$ is locally numbered $\overline{u}_0$ as shown in figure 4.1a)

$$\begin{aligned} \mathrm{h}(u^+, u^-, \vec{\mathbf{n}})_i = &\, \mathrm{h}(\overline{u}_i, \overline{u}_0, \vec{\mathbf{n}}_i) \\ = &\, \frac{1}{2}(f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i)) \\ &- \frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0) \end{aligned} \tag{4.67}$$

In this formula, $\vec{\mathbf{n}}_i = \int_{\mathbf{R}'_{i-1/2}}^{\mathbf{R}'_{i+1/2}}$ for any simple path. By summing over all edges of the control volume, the entire scheme for c_j is written

$$\begin{aligned} \frac{\partial}{\partial t}\int_{c_j} \overline{u}_0\, da + &\sum_{i=1}^{d(c_j)} \frac{1}{2}(f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i)) \\ &- \sum_{i=1}^{d(c_j)} \frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0) = 0 \end{aligned} \tag{4.68}$$

It is not difficult to prove stability and monotonicity of this scheme.

Monotonicity and Stability

Recall that the flux function was constructed from a mean value linearization such that

$$f(u_i, \vec{n}_i) - f(u_0, \vec{n}_i) = a(\tilde{u}_i, \vec{n}_i)(u_i - u_0) \quad (4.69)$$

with $\tilde{u}_i = \theta u_0 + (1-\theta)u_i$, $\theta \in [0,1]$. This permits regrouping terms into the following form:

$$\frac{\partial}{\partial t}\int_{c_j} \overline{u}_0 \, da + \sum_{i=1}^{d(c_j)} f(u_0, \vec{n}_i) + \sum_{i=1}^{d(c_j)} \frac{1}{2}\left(a(\tilde{u}_i, \vec{n}_i) - |a(\tilde{u}_i, \vec{n}_i)|\right)(\overline{u}_i - \overline{u}_0) = 0 \quad (4.70)$$

For any closed control volume, we have that

$$\sum_{i=1}^{d(c_j)} f(\overline{u}_0, \vec{n}_i) = 0$$

Combining the remaining terms yields a final form for analysis ($a = a^+ + a^-, |a| = a^+ - a^-$):

$$\frac{\partial}{\partial t}\int_{c_j} \overline{u}_0 \, da + \sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^-(\overline{u}_i - \overline{u}_0) = 0 \quad (4.71)$$

To verify the monotonicity of the scheme at steady state, set the time term to zero and solve for $\overline{u}_0$.

$$\overline{u}_0 = \frac{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^- \overline{u}_i}{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^-} = \sum_{i=1}^{d(c_j)} \alpha_i \overline{u}_i \quad (4.72)$$

All weights α_i are positive and sum to unity. The scheme is monotone since $\overline{u}_0$ is a positive weighted average of all neighbors. This implies a *maximum principle* since $\overline{u}_0$ is bounded from above and below by the maximum and minimum of neighboring values (and itself), $\overline{u}_{max}$ and $\overline{u}_{min}$.

$$\overline{u}_{min} \le \overline{u}_0 \le \overline{u}_{max} \quad (4.73)$$

For explicit time stepping, a CFL-like condition is obtained for monotonicity. For Euler explicit time stepping, we have the time approximation,

$$\frac{\partial}{\partial t}\frac{1}{A_c}\int_c \overline{u}_0 \, da \approx \frac{\overline{u}_0^{n+1} - \overline{u}_0^n}{\Delta t}$$

which results in the following scheme:

$$\begin{aligned}\overline{u}_0^{n+1} &= \overline{u}_0^n - \frac{\Delta t}{A_c}\sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^-(\overline{u}_i^n - \overline{u}_0^n) \\ &= \sum_{i=0}^{d(c_j)} \alpha_i \overline{u}_i^n\end{aligned} \quad (4.74)$$

It should be clear that coefficients in (4.74) sum to unity. To prove monotonicity in time and space, it is sufficient to show positivity of coefficients. By inspection we have that $\alpha_i \ge 0 \;\; \forall\, i > 0$. To guarantee monotonicity requires that $\alpha_0 \ge 0$.

$$\alpha_0 = 1 + \frac{\Delta t}{A_c}\sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^- \ge 0 \quad (4.75)$$

Thus, a CFL-like condition is obtained which insures monotonicity and stability.

$$\Delta t \le -\frac{A_c}{\sum_{i=1}^{d(c_j)} a(\tilde{u}_i, \vec{n}_i)^-} \quad (4.76)$$

Note that in one dimension, this number corresponds to the conventional CFL number. In multiple space dimensions, this inequality is sufficient but not necessary for stability. In practice somewhat larger timestep values may be used.

Conclusion: *The upwind algorithm (4.68) using piecewise constant data satisfies a discrete maximum principle for general unstructured meshes.*

4.7 Upwind Advection Schemes with Linear ($k = 1$) Reconstruction

In this section, we consider advection schemes based on linear reconstruction. The process of linear reconstruction in one dimension is depicted in figure 4.9.

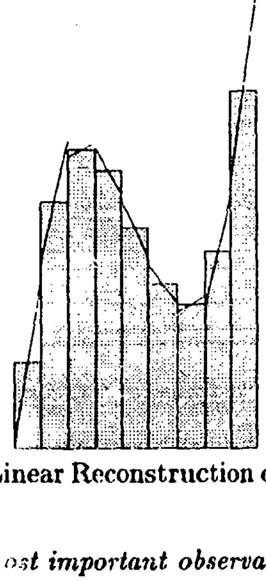


Figure 4.9 Linear Reconstruction of cell-averaged data.

One of the most important observations concerning linear reconstruction is that we can dispense with the notion of cell averages as unknowns by reinterpreting the unknowns as pointwise values of the solution sampled at the centroid (midpoint in 1-D) of the control volume. This well known result greatly simplifies schemes based on linear reconstruction. The linear reconstruction in each interval shown in figure 4.9 was obtained by a simple central-difference formula given point values of the solution at the midpoint of each interval.

In section 4.3, results for the Ringleb flow with linear reconstruction were presented. The reconstruction strategy presented there satisfies all the design requirements of the reconstruction operator. For linear reconstruction, simpler formulations are possible which exploit the edge data structure. Several of these reconstruction schemes are given below. Note that for steady-state computations, conservation of the mean in the data reconstruction is not necessary. The implication of violating this conservation is that a *nondiagonal* mass matrix appears in the time integral. Since time derivatives vanish at steady-state, the effect of this mass matrix vanishes at steady-state. The reconstruction schemes presented below assume that solution variables are placed at the vertices of the mesh, which may not be at the precise centroid of the control volume, thus violating conservation of the mean. The schemes can all be implemented using an edge data structure and satisfy k-exactness for linear functions.

4.7a Green-Gauss Reconstruction

This reconstruction exploits the gradient calculation (4.19) studied earlier in section 4.3:

$$(\nabla u)_{v_0} = \frac{1}{A_0} \sum_{i \in \mathcal{I}_0} \frac{1}{2}(u_i + u_0)\vec{\mathbf{n}}_{0i} \qquad (4.77)$$

where $\vec{\mathbf{n}}_{0i}$ is the vector normal associated with the edge $e(v_0, v_i)$. This approximation extends naturally to three dimensions, see Barth [43].

4.7b Linear Least-Squares (L_2) Reconstruction

To derive this reconstruction technique, consider a vertex v_0 and suppose that the solution varies linearly over the support of adjacent neighbors of the mesh. In this case, the change in vertex values of the solution along an edge $e(v_i, v_0)$ can be calculated by

$$(\nabla u^h)_0 \cdot (\mathbf{R}_i - \mathbf{R}_0) = u_i - u_0 \qquad (4.78)$$

This equation represents the scaled projection of the gradient along the edge $e(v_i, v_0)$. A similar equation could be written for all incident edges subject to an arbitrary weighting factor. The result is the following matrix equation, shown here in three dimensions:

$$\begin{bmatrix} w_1\Delta x_1 & w_1\Delta y_1 & w_1\Delta z_1 \\ \vdots & \vdots & \vdots \\ w_n\Delta x_n & w_n\Delta y_n & w_n\Delta z_n \end{bmatrix} \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix} = \begin{pmatrix} w_1(u_1 - u_0) \\ \vdots \\ w_n(u_n - u_0) \end{pmatrix} \qquad (4.79)$$

or in symbolic form $\mathcal{L}\,\nabla u = \mathbf{f}$ where

$$\mathcal{L} = \left[\, \vec{\mathbf{L}}_1 \quad \vec{\mathbf{L}}_2 \quad \vec{\mathbf{L}}_3 \,\right] \qquad (4.80)$$

in three dimensions. Exact calculation of gradients for linear u is guaranteed if any three row vectors $w_i(\mathbf{R}_i - \mathbf{R}_0)$ span all of 3 space. This implies linear independence of $\vec{\mathbf{L}}_1$, $\vec{\mathbf{L}}_2$, and $\vec{\mathbf{L}}_3$. The system can then be solved via a Gram-Schmidt process, i.e.,

$$\begin{bmatrix} \vec{\mathbf{V}}_1 \\ \vec{\mathbf{V}}_2 \\ \vec{\mathbf{V}}_3 \end{bmatrix} \left[\, \vec{\mathbf{L}}_1 \quad \vec{\mathbf{L}}_2 \quad \vec{\mathbf{L}}_3 \,\right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (4.81)$$

The row vectors $\vec{\mathbf{V}}_i$ are given by $\vec{\mathbf{V}}_i = \frac{\vec{\mathbf{U}}_i}{(\vec{\mathbf{L}}_i \cdot \vec{\mathbf{U}}_i)}$ where

$$\begin{aligned} \vec{\mathbf{U}}_1 =& (l_{33}l_{22} - l_{23}l_{23})\vec{\mathbf{L}}_1 - (l_{33}l_{12} - l_{23}l_{13})\vec{\mathbf{L}}_2 \\ & -(l_{22}l_{13} - l_{23}l_{12})\vec{\mathbf{L}}_3 \\ \vec{\mathbf{U}}_2 =& (l_{33}l_{11} - l_{13}l_{13})\vec{\mathbf{L}}_2 - (l_{33}l_{12} - l_{13}l_{23})\vec{\mathbf{L}}_1 \\ & -(l_{11}l_{23} - l_{13}l_{12})\vec{\mathbf{L}}_3 \\ \vec{\mathbf{U}}_3 =& (l_{11}l_{22} - l_{12}l_{12})\vec{\mathbf{L}}_3 - (l_{22}l_{13} - l_{12}l_{23})\vec{\mathbf{L}}_1 \\ & -(l_{11}l_{23} - l_{12}l_{13})\vec{\mathbf{L}}_2 \end{aligned}$$

and $l_{ij} = (\vec{\mathbf{L}}_i \cdot \vec{\mathbf{L}}_j)$.

Note that reconstruction of N independent variables in $\mathbf{R}^d$ [illegible] lies $\binom{d+1}{2} + d\,N$ inner product sums. Since only $d\,N$ of these sums involves the solution variables themselves, the remaining sums could be precalculated and stored in computer memory. This makes the present scheme competitive with the Green-Gauss reconstruction. Using the edge data structure, the calculation of inner product sums can be calculated for *arbitrary* combinations of polyhedral cells. In all cases linear functions are reconstructed exactly. We demonstrate this idea by example:

```
For $k = 1, n(e)$   ! Loop through edges of mesh
  $j_1 = e^{-1}(k, 1)$   ! Pointer to edge origin
  $j_2 = e^{-1}(k, 2)$   ! Pointer to edge destination
  $dx = w(k) \cdot (x(j_2) - x(j_1))$   ! Weighted $\Delta x$
  $dy = w(k) \cdot (y(j_2) - y(j_1))$   ! Weighted $\Delta y$
  $l_{11}(j_1) = l_{11}(j_1) + dx \cdot dx$   ! $l_{11}$ orig sum
  $l_{11}(j_2) = l_{11}(j_2) + dx \cdot dx$   ! $l_{11}$ dest sum
  $l_{12}(j_1) = l_{12}(j_1) + dx \cdot dy$   ! $l_{12}$ orig sum
  $l_{12}(j_2) = l_{12}(j_2) + dx \cdot dy$   ! $l_{12}$ dest sum
   .
   .
   .
  $du = w(k) \cdot (u(j_2) - u(j_1))$   ! Weighted $\Delta u$
  $lu_3(j_1) = lu_3(j_1) + dz \cdot du$   ! $lu_3$ orig sum
```

```
$lu_3(j_2) = lu_3(j_2) + dz \cdot du$    ! $lu_3$ dest sum
Endfor
```

This formulation provides freedom in the choice of weighting coefficients, w_i. These weighting coefficients can be a function of the geometry and/or solution. Classical approximations in one dimension can be recovered by choosing geometrical weights of the form $w_i = 1./|\mathbf{R}_i - \mathbf{R}_0|^t$ for values of $t = 0, 1, 2$. The L_2 gradient calculation technique is optimal in a weighted least squares sense and determines gradient coefficients with least sensitivity to Gaussian noise. This is an important property when dealing with highly distorted (stretched) meshes.

4.7c Data Dependent Reconstruction

Both the Green-Gauss and L_2 gradient calculation techniques can be generalized to include data dependent (i.e. solution dependent) weights. In the case of Green-Gauss formulation, the sum

$$\sum_{i\in\mathcal{I}_0} \frac{1}{2}(u_0 + u_i)\vec{\mathbf{n}}_{0i}$$

is replaced by

$$\sum_{i\in\mathcal{I}_0} p^-_{0i}\frac{1}{2}(u_0+u_i)\vec{\mathbf{n}}_{0i}+p^+_{0i}\frac{1}{2}\left((\nabla u)_0 \cdot (\mathbf{R}_i - \mathbf{R}_0)\right)\vec{\mathbf{n}}_{0i} \tag{4.82}$$

If the $p^\pm_{0i}$ are chosen such that $p^-_{0i} + p^+_{0i} = 1$ then the gradient calculation is exact whenever the solution varies linearly over the support. In two space dimensions, equation (4.82) implies the solution of a linear 2×2 system of the form

$$\begin{bmatrix} A_0 - m_{xx} & -m_{xy} \\ -m_{yx} & A_0 - m_{yy}\end{bmatrix}\begin{pmatrix}u_x\\ u_y\end{pmatrix} = \sum_{i\in\mathcal{I}_0} p^-_{0i}\frac{1}{2}(u_0+u_i)\vec{\mathbf{n}}_{0i}$$

where

$$m_{xx} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta x_i n_{x_i}, \quad m_{yy} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta y_i n_{y_i}$$

$$m_{xy} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta x_i n_{y_i}, \quad m_{yx} = \sum_{i\in\mathcal{I}_0} p^+_{0i}\Delta y_i n_{x_i}$$

Care must be exercised in the selection of $p^\pm$ in order that the system be invertible. This is similar to the spanning space requirement of the L_2 gradient calculation technique.

4.7d Monotonicity Enforcement

When solution discontinuites and steep gradients as present, additional steps must be taken to prevent oscillations from developing in the numerical solution. One way to do this was pioneered by van Leer [35] in the late 1970's. The basic idea is to take the reconstructed piecewise polynomials and enforce strict monotonicity in the reconstruction. Monotonicity in this context should be interpreted to mean that the value of the reconstructed polynomial does not exceed the minimum and maximum of neighboring cell averages. In other words, the final reconstruction must guarantee that no new extrema have been created. This will be referred to as 'monotonicity property 1.' When a new extremum is produced, the slope of the reconstruction in that interval is reduced until monotonicity is restored. This implies that at a local minimum or maximum in the cell averaged data the slope in 1-D is *always* reduced to zero, see for example figure 4.10.

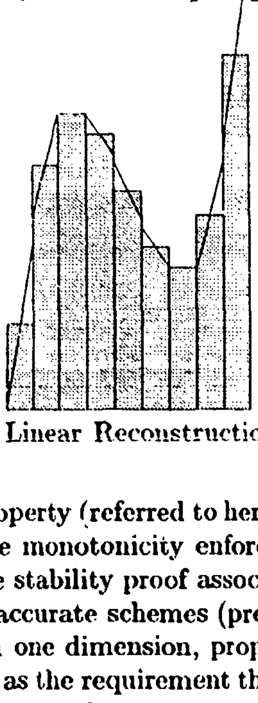


Figure 4.10 Linear Reconstruction with monotone limiting.

Another property (referred to hereafter as 'property 2') of the monotonicity enforcement is motivated by the stability proof associated with the higher order accurate schemes (presented in section 4.7e). In one dimension, property 2 can be characterized as the requirement that the new reconstruction not produce a reconstructed solution variation, $\int |du|$, which is larger than the piecewise constant value. If property 2 is violated then the slopes must be reduced until the solution variation is satisfied. This situation is depicted in figure 4.11. For arbitrary unstructured grids, a *sufficient* condition is that the differences in the extrapolated states at a cell interface quadrature point be of the same sign as the difference in the piecewise constant values, i.e.

$$\frac{u^+ - u^-}{\bar{u}^+ - \bar{u}^-} \geq 0, \quad \text{(property 2)}$$

when combined with property 1, the following inequality exists:

$$1 \geq \frac{u^+ - u^-}{\bar{u}^+ - \bar{u}^-} \geq 0 \tag{4.83}$$

This inequality is crucial in the stability proof given below.

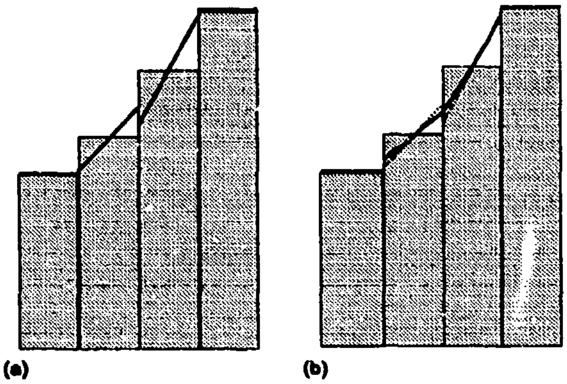


Figure 4.11 (a) Reconstruction profile with increased variation violating monotonicity property 2. (b) Profile after modification to satisfy monotonicity property 2.

In Barth and Jespersen [40], we gave a simple recipe for invoking property 1. Consider writing the linearly reconstructed data in the following form:

$$u^k(x,y)_j = \overline{u}_j + \nabla u^k_{c_j} \cdot (\mathbf{R} - \mathbf{R}_j) \qquad (4.84a)$$

Now consider a "limited" form of this piecewise linear distribution.

$$u^k(x,y)_j = \overline{u}_j + \Phi_j \nabla u^k_{c_j} \cdot (\mathbf{R} - \mathbf{R}_j) \qquad (4.84b)$$

The idea is to find the largest admissible Φ_j while invoking a monotonicity principle that values of the linearly reconstructed function must not exceed the maximum and minimum of neighboring centroid values (including the centroid value in c_j). To do this, first compute

$$u_j^{min} = \min(\overline{u}_j, \overline{u}_{neighbors})$$

and

$$u_j^{max} = \max(\overline{u}_j, \overline{u}_{neighbors})$$

then require that

$$u_j^{min} \leq u(x,y)^k_j \leq u_j^{max} \qquad (4.85)$$

For linear reconstructions, extrema in $u(x,y)^k_j$ occur at the vertices of the control volume and sufficient conditions for (4.85) can be easily obtained. For each vertex of the cell compute $u_i = u^k(x_i, y_i)_j$, $i = 1, N_{c_j}$ to determine the limited value, ϕ_i, which satisfies (4.84):

$$\phi_i = \begin{cases} \min(1, \frac{u_j^{max} - \overline{u}_j}{u_i - \overline{u}_j}), & \text{if } u_i - \overline{u}_j > 0 \\ \min(1, \frac{u_j^{min} - \overline{u}_j}{u_i - \overline{u}_j}), & \text{if } u_i - \overline{u}_j < 0 \\ 1 & \text{if } u_i - \overline{u}_j = 0 \end{cases}$$

with $\Phi_j = \min(\phi_1, \phi_2, \phi_3, ..., \phi_{N_{c_j}})$. In practice, the reconstructed polynomial may be calculated at the flux quadrature points instead of the vertices of the control volume with a negligible degradation in monotonicity. In the implementation of property 2, we prefer a "symmetric" reduction of slopes. In other words, at interfaces violating property 2, both of the two cells sharing that interface reduce their slope until (4.83) is satisfied.

When the above procedures are combined with the flux function given earlier (4.58),

$$\begin{aligned} h(u^+, u^-, \mathbf{n}) =& \frac{1}{2}\left(f(u^+, \mathbf{n}) + f(u^-, \mathbf{n})\right) \\ &- \frac{1}{2}|a(\tilde{u}, \mathbf{n})|\left(u^+ - u^-\right) \end{aligned} \qquad (4.58)$$

the resulting scheme has very good shock resolving characteristics. To demonstrate this fact, we consider the scalar nonlinear hyperbolic problem suggested by Struijs, Deconinck, *et al* [44]. The equation is a multidimensional form of Burger's equation.

$$u_t + (u^2/2)_x + u_y = 0$$

We solve the equation in a square region $[0, 1.5] \times [0, 1.5]$ with boundary conditions: $u(x,0) = 1.5 - 2x$, $x \leq 1$, $u(x,0) = -.5$, $x > 1$, $u(0,y) = 1.5$, and $u(1.5, y) = -.5$. Figures 4.12 and 4.13 show carpet plots and contours of the solution on regular and irregular meshes.

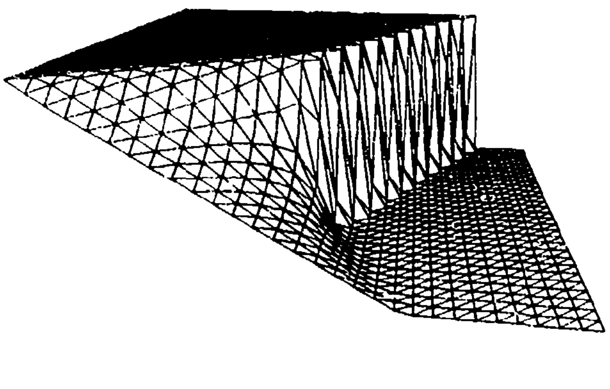


Figure 4.12a Carpet plot of Burger's equation solution on regular mesh.

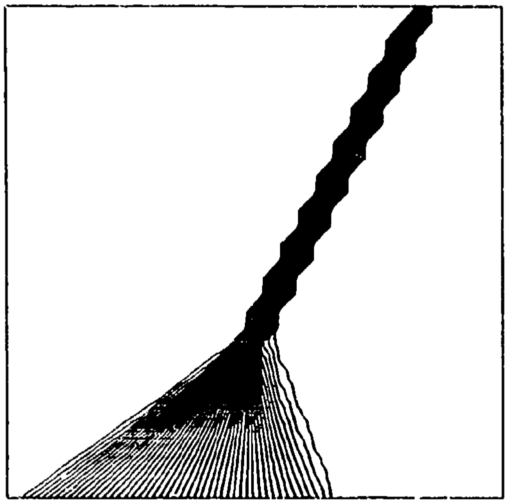


Figure 4.12b Solution Contours on regular mesh.

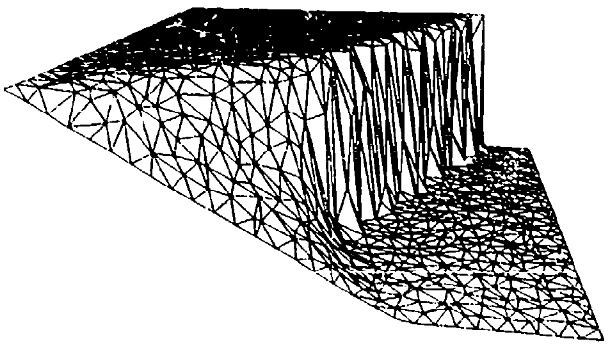


Figure 4.13a Carpet plot of Burger's equation solution on irregular mesh.

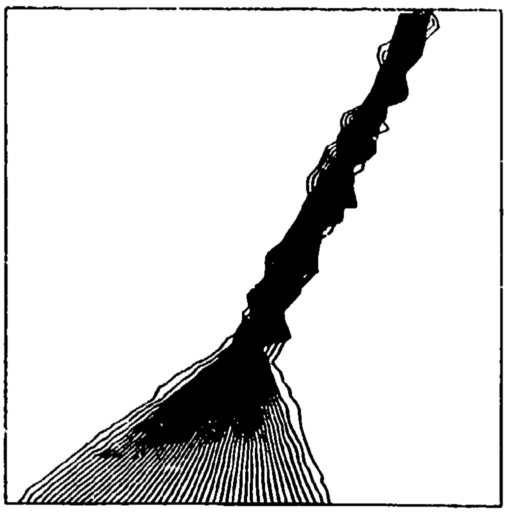


Figure 4.13b Solution contours.

Note that the carpet plots indicate that the numerical solution on both meshes is monotone. Even so, most people would prefer the solution on the regular mesh. This is an unavoidable consequence of irregular meshes. The only remedy appears to be mesh adaptation. Sim'ar results for the Euler equations will be shown on irregular meshes in a future section.

<u>4.7e Stability Analysis via Energy Methods</u>

Consider once again the local mesh shown in figure 4.1a with local index about a vertex v_0. In the analysis performed below, we consider energy stability of schemes of the following form

$$\frac{\partial}{\partial t}\overline{u}_0 A_0 = -\sum_{i=1}^{d(c_j)} \mathbf{h}(u^+, u^-, \vec{\mathbf{n}})_{0i} \qquad (4.86)$$

using linear reconstruction with limiting. Note that in this analysis all boundary effects will be ignored. In section 4.5, stability of the first order upwind scheme was proven using monotonicity analysis. Before considering the higher order schemes, we briefly digress to show stability of the first order upwind scheme using energy methods. Using the same techniques, energy stability of the high order schemes with reconstruction and limiting will be shown.

<u>Energy Analysis for the $k = 0$ scheme</u>

In this case, the flux takes the simple form and the scheme for a single vertex v_0 can be written as indicated below

$$\underbrace{(A_0\overline{u}_0)}_{\mathcal{D}\mathbf{u}}{}_t + \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}(f(\overline{u}_0, \vec{\mathbf{n}}_i) + f(\overline{u}_i, \vec{\mathbf{n}}_i))}_{\mathcal{L}_a\mathbf{u}} \\ - \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}|a(\tilde{u}_i, \vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0)}_{\mathcal{L}_d\mathbf{u}} = 0 \qquad (4.87)$$

or in symbolic operator form, where $\mathbf{u}$ denotes the solution vector, i.e. $\mathbf{u} = [\overline{u}_1, \overline{u}_2, \overline{u}_3, ..., \overline{u}_N]^T$. In this symbolic form, the scheme is written as

$$(D\mathbf{u})_t + \mathcal{L}_a\mathbf{u} - \mathcal{L}_d\mathbf{u} = 0 \qquad (4.88)$$

where D is a positive diagonal matrix cont ining the area of each control volume. $\mathcal{L}_a$ and $\mathcal{L}_s$ represent the advective and diffusive operators in this linear scheme. The energy of the system (1.88) is given by the following equation:

$$(\mathbf{u}^T D_0 \mathbf{u})_t + \mathbf{u}^T(\mathcal{L}_a + \mathcal{L}_a^T)\,\mathbf{u} - \mathbf{u}^T(\mathcal{L}_d + \mathcal{L}_d^T)\,\mathbf{u} = 0 \qquad (4.89)$$

It is a straightforward exercise to show that in the linear case, $\mathcal{L}_a$ and $\mathcal{L}_a^T$ are skew-symmetric (isoenergetic) operators, hence

$$\mathbf{u}^T\left(\mathcal{L}_a + \mathcal{L}_a^T\right)\mathbf{u} = 0.$$

The diffusive operator $\mathcal{L}_d$ is symmetric which reduces the energy equation to the following form:

$$\frac{1}{2}\left(\mathbf{u}^T D_0 \mathbf{u}\right)_t - \mathbf{u}^T\mathcal{L}_d\mathbf{u} = 0 \qquad (4.90)$$

From symmetry and application of the eigenvalue circle theorem, it is easily shown that $\mathcal{L}_d$ is a symmetric, negative semi-definite matrix operator which implies that

$$\mathbf{u}^T\mathcal{L}_d\mathbf{u} \le 0$$

for all $\mathbf{u}$. This establishes that the scheme is energy stable since

$$\left(\mathbf{u}^T D_0 \mathbf{u}\right)_t \le 0$$

Energy Analysis for the $k = 1$ scheme

We now consider the advection scheme with linear reconstruction. The interface states for the edge of the control volume separating cells c_0 and c_i are denoted by u_0^+ and u_i^-, respectively. The scheme is written in the familiar form:

$$\underbrace{(A_0\overline{u}_0)_t}_{\mathcal{D}\mathbf{u}} + \underbrace{\sum_{i=1}^{d(c_j)} \frac{1}{2}\left(f(u_0^+,\vec{\mathbf{n}}_i) + f(u_i^-,\vec{\mathbf{n}}_i)\right)}_{\mathcal{L}_a\mathbf{u}}$$
$$\underbrace{-\sum_{i=1}^{d(c_j)} \frac{1}{2}|a(\tilde{u}_i,\vec{\mathbf{n}}_i)|\,(u_i^- - u_0^+)}_{\mathcal{L}_d\mathbf{u}} = 0 \qquad (4.91)$$

Consider rewriting equation (4.91) using the identity

$$u_i^- - u_0^+ = \left(\frac{u_i^- - u_0^+}{\overline{u}_i - \overline{u}_0}\right)(\overline{u}_i - \overline{u}_0) = \psi(\overline{u}_i - \overline{u}_0)$$

which tacitly assumes that the ratio exists. Monotonicity properties 1 and 2 guarantee that $\psi \in [0,1]$. Thus, equation (4.91) is rewritten in the nonlinear form:

$$(A_0\overline{u}_0)_t + \sum_{i=1}^{d(c_j)} \frac{1}{2}\left(f(u_0^+,\vec{\mathbf{n}}_i) + f(u_i^-,\vec{\mathbf{n}}_i)\right)$$
$$-\sum_{i=1}^{d(c_j)} \frac{1}{2}\psi_i|a(\tilde{u}_i,\vec{\mathbf{n}}_i)|(\overline{u}_i - \overline{u}_0) = 0$$

From symmetry and the eigenvalue circle theorem we have that

$$\mathbf{u}^T\mathcal{L}_d\mathbf{u} \le 0 \qquad (4.92)$$

It remains to be shown that the advection operator $\mathcal{L}_a$ is either isoenergetic (in the linear case) or decays energy in the system. Not all extrapolation formulas guarantee that this is true. A full discussion of this topic is beyond the scope of these notes and is a subject of current research. Note that in reference [45], we indicated a preference for a standard Galerkin discretization of $\mathcal{L}_a$. Since this operator is isoenergetic, when combined with the diffusion operator described above, the entire scheme is provably stable in an energy norm.

4.8 Maximum Principles and the Delaunay Triangulation

The edge formulas presented earlier not only provide an efficient procedure for calculating quantities such as the gradient and divergence, but also provide certain theoretical results which are difficult to ascertain otherwise. For example, Ciarlet and Raviart [46] consider Galerkin schemes for solving elliptic equations using linear finite-elements. They derive sufficient conditions for the existence of a discrete maximum principle for Laplace's equation if all angles in the triangulation are less than $\pi/2 - \epsilon$ for some positive ϵ. Using the edge formulas derived in section 4.3, sufficient and necessary conditions can be derived for a discrete maximum principle which are quite different from the Ciarlet result. A brief outline of the proof is given below.

Example: Derive conditions for a discrete maximum principle using a Galerkin approximation with linear elements.

Using a reduced form of (4.43), the canonical edge formula for the discrete Laplacian operator is given by

$$\int_{\Omega_0} \phi\Delta u^h da = \mathcal{L}(u^h)_{v_0} =$$
$$\sum_{i\in\mathcal{I}_0} \frac{1}{2}[\cot\mathrm{an}(\alpha_{L_i}) + \cot\mathrm{an}(\alpha_{R_i})]\,(u_i - u_0) \qquad (4.93)$$

where the angles α_{L_i} and α_{R_i} are depicted below.

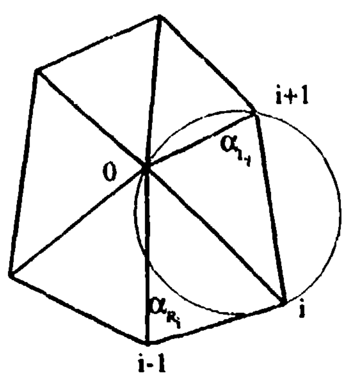


Figure 4.14 Circumcircle test for adjacent triangles.

It is well known that a discrete maximum principle exists for arbitrary point distributions and boundary data if and only if the discrete operator is a nonnegative operator, i.e., if

$$\mathcal{L}(u^h)_{v_0} = \sum_{i\in\mathcal{I}_0} w_i u_i^h \qquad (4.94)$$

and

$$w_0 < 0,\ w_i \geq 0, i > 0,\ w_0 + \sum_{i\in\mathcal{I}_0} w_i = 0 \quad (4.95)$$

for any interior vertex v_0. For schemes of the form

$$\mathcal{L}(u^h)_{v_0} = \sum_{i\in\mathcal{I}_0} W_i(u_i^h - u_0^h) \qquad (4.96)$$

nonnegativity requires that $W_i \geq 0$ for all $i \in \mathcal{I}_0$. This guarantees a maximum principle. Equating equation (4.96) to zero, we obtain

$$u_0^h = \frac{\sum_{i\in\mathcal{I}_0} W_i u_i^h}{\sum_{i\in\mathcal{I}_0} W_i} \qquad (4.97)$$

and therefore

$$\min(u_1^h, u_2^h, ..., u_{d_0}^h) \leq u_0^h \leq \max(u_1^h, u_2^h, ..., u_{d_0}^h)$$

A natural question to be addressed concerns the existence and uniqueness of triangulations of an arbitrary point set such that (4.93) guarantees a discrete maximum principle. In two dimensions a unique triangulation always exists. The main result is summarized in the following theorem:

The discrete Laplacian operator (4.93) exhibits a discrete maximum principle for arbitrary point sets in two space dimensions iff the triangulation of these points is a Delaunay triangulation.

The key elements of the proof are given below: Rearrangement of the weights appearing in (4.93) yields

$$\begin{aligned} W_i &= \frac{1}{2}\left[\text{cotan}(\alpha_{L_i}) + \text{cotan}(\alpha_{R_i})\right] \\ &= \frac{1}{2}\left[\frac{\cos(\alpha_{L_i})}{\sin(\alpha_{L_i})} + \frac{\cos(\alpha_{R_i})}{\sin(\alpha_{R_i})}\right] \\ &= \frac{1}{2}\left[\frac{\sin(\alpha_{L_i} + \alpha_{R_i})}{\sin(\alpha_{L_i})\sin(\alpha_{R_i})}\right] \end{aligned} \qquad (4.98)$$

Since $\alpha_{L_i} < \pi$, $\alpha_{R_i} < \pi$, the denominator is always positive and nonnegativity requires that $\alpha_{L_i} + \alpha_{R_i} \leq \pi$. Some trigonometry reveals that for the configuration of figure 4.14 with circumcircle passing through $\{v_0, v_i, v_{i+1}\}$ the sum $\alpha_{R_i} + \alpha_{L_i}$ depends on the location of v_{i-1} with respect to the circumcircle in the following way:

$$\begin{aligned} &\alpha_{R_i} + \alpha_{L_i} < \pi, && v_{i-1} \text{ exterior} \\ &\alpha_{R_i} + \alpha_{L_i} > \pi, && v_{i-1} \text{ interior} \\ &\alpha_{R_i} + \alpha_{L_i} = \pi, && v_{i-1} \text{ cocircular} \end{aligned} \qquad (4.99)$$

Also note that we could have considered the circumcircle passing through $\{v_0, v_i, v_{i-1}\}$ with similar results for v_{i+1}. The condition of nonnegativity implies a circumcircle condition for all pairs of adjacent triangles whereby the circumcircle passing through either triangle cannot contain the fourth point. This is precisely the unique characterization of the Delaunay triangulation which would complete the proof.

Keep in mind that from equation (4.98) we have that $\text{cotan}(\alpha) \geq 0$ if $\alpha \leq \pi/2$. Therefore a sufficient but not necessary condition for nonnegativity (and a Delaunay triangulation) is that all angles of the mesh be less than or equal to $\pi/2$. This is a standard result in finite element theory and applies in two or more space dimensions. The construction of nonnegative operators has important implications with respect to the diagonal dominance of implicit schemes, the eigenvalue spectrum of the discrete operator, stability of relaxation schemes, etc.

We can ask if the result concerning Delaunay triangulation and the maximum principle extends to three space dimensions. As we will show, the answer is no. Section 4.3d gives the corresponding edge formulas for Hessian and Laplacian discretizations in 3-D. The resulting formula for the three dimensional Laplacian is

$$\int_{V_{v_0}} \phi^h \nabla^2 u^h \, dv = \sum_{i\in\mathcal{I}_0} W_i(u_i - u_0) \quad (4.100)$$

$$W_i = \frac{1}{6}\sum_{k=1}^{d(v_0,v_i)} |\Delta\mathbf{R}_{k+1/2}| \cot\text{an}(\alpha_{k+1/2}). \tag{4.101}$$

In this formula V_{Γ_0} is the volume formed by the union of all tetrahedra that share vertex v_0. $\mathcal{I}_0$ is the set of indices of all adjacent neighbors of v_0 connected by incident edges, k a local cyclic index describing the associated vertices which form a polygon of degree $d(v_0, v_i)$ surrounding the edge $e(v_0, v_i)$, $\alpha_{k+1/2}$ is the face angle between the two faces associated with $\vec{\mathbf{s}}_{k+1/2}$ and $\vec{\mathbf{s}}'_{k+1/2}$ which share the edge $e(v_k, v_{k+1})$ and $|\Delta\mathbf{R}_{k+1/2}|$ is the magnitude of the edge (see figure below).

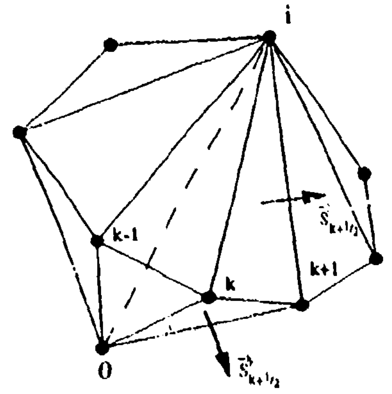


Figure 4.15 Set of tetrahedra sharing interior edge $e(v_0, v_i)$ with local cyclic index k.

A maximum principle is guaranteed if all $W_i \geq 0$. We now will procede to describe a valid Delaunay triangulation with one or more $W_i < 0$. It will suffice to consider the Delaunay triangulation of N points in which a single point v_0 lies interior to the triangulation and the remaining $N-1$ points describe vertices of boundary faces which completely cover the convex hull of the point set.

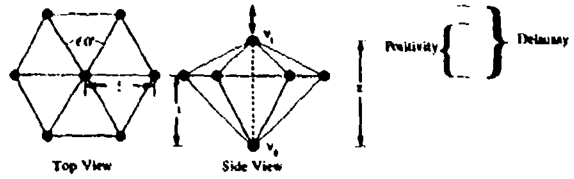


Figure 4.16 Subset of 3-D Delaunay Triangulation which does not maintain nonnegativity.

Consider a subset of the N vertices, in particular consider an interior edge incident to v_0 connecting to v_i as shown in figure 4.16 by the dashed line segment and all neighbors adjacent to v_i on the hull of the point set. In this experiment we consider the height of the interior edge, z, as a free parameter. Although it will not be proven here, the remaining $N-8$ points can be placed without conflicting with any of the conclusions obtained for looking at the subset.

It is known that a necessary and sufficient condition for the 3-D Delaunay triangulation is that the circumsphere passing through the vertices of any tetrahedron must be point free [21]; that is to say that no other point of the triangulation can lie interior to this sphere. Furthermore a property of locality exists [21] so that we need only inspect adjacent tetrahedra for the satisfaction of the circumsphere test. For the configuration of points shown in figure 4.16, convexity of the point cloud constrains $z \geq 1$ and the satisfaction of the circumsphere test requires that $z \leq 2$.

$$1 \leq z \leq 2 \quad \text{(Delaunay Triangulation)}$$

From (2.13) we find that $W_i \geq 0$ if and only if $z < 7/4$.

$$1 \leq z \leq \frac{7}{4}, \quad \text{(Nonnegativity)}$$

This indicates that for $7/4 < z \leq 2$ we have a valid Delaunay triangulation which does not satisfy a discrete maximum principle. In fact, the Delaunay triangulation of 400 points randomly distributed in the unit cube revealed that approximately 25% of the interior edge weights were of the wrong sign (negative).

Keep in mind that from (4.101) we can obtain a sufficient but not necessary condition for nonnegativity that all face angles be less than or equal to $\pi/2$.

The formulas for the prototypical viscous term $\nabla \cdot \mu\nabla u$ are only slightly more complicated than the Laplacian formulas. In 2-D we have the following weights

$$W_i = \frac{1}{2}\left[\overline{\mu}_{L_i}\cot\text{an}(\alpha_{L_i}) + \overline{\mu}_{R_i}\cot\text{an}(\alpha_{R_i})\right] \tag{4.102}$$

or in 3-D

$$W_i = \frac{1}{6}\sum_{k=1}^{d(v_0,v_i)} \overline{\mu}_{k+1/2}|\Delta\mathbf{R}_{k+1/2}| \cot\text{an}(\alpha_{k+1/2}) \tag{4.103}$$

where $\overline{\mu}$ is the average value of μ in the specified simplex. Since μ and $\overline{\mu}$ are always assumed positive quantities, we have the following theorem:

A discrete maximum principle associated with the discretization of $\nabla \cdot \mu\nabla u$ with weights given by (4.102) and (4.103) is guaranteed iff $W_i \geq 0$ for all interior edges of the mesh. A sufficient but not necessary condition is that all angles (2-D) or faces angles (3-D) be less than or equal to $\pi/2$.

The proof follows immediately from nonnegativity of (4.102) and (4.103). The sufficient but not necessary condition is a minor extension of the result by Ciarlet [46].

5.0 Finite-Volume Solvers for the Euler Equations

In this section, we consider the extension of upwind scalar advection schemes to the Euler equations of gasdynamics. As we will see, the changes are relatively minor since most of the difficult work has already been done in designing the scalar scheme.

5.1 Euler Equations in Integral Form

The physical laws concerning the conservation of mass, momentum, and energy for an arbitrary region Ω can be written in the following integral form:

Conservation of Mass

$$\frac{\partial}{\partial t}\int_{\Omega} \rho \, da + \int_{\partial\Omega} \rho(\mathbf{V}\cdot\mathbf{n}) \, dl = 0 \qquad (5.1)$$

Conservation of Momentum

$$\frac{\partial}{\partial t}\int_{\Omega} \rho\mathbf{V} \, da + \int_{\partial\Omega} \rho\mathbf{V}(\mathbf{V}\cdot\mathbf{n}) \, dl + \int_{\partial\Omega} p\mathbf{n} \, dl = 0 \qquad (5.2)$$

Conservation of Energy

$$\frac{\partial}{\partial t}\int_{\Omega} E \, da + \int_{\partial\Omega} (E+p)(\mathbf{V}\cdot\mathbf{n}) \, dl = 0 \qquad (5.3)$$

In these equations $\rho, \mathbf{V}, p$, and E are the density, velocity, pressure, and total energy of the fluid. The system is closed by introducing a thermodynamical equation of state for a perfect gas:

$$p = (\gamma - 1)(E - \frac{1}{2}\rho(\mathbf{V}\cdot\mathbf{V})) \qquad (5.4)$$

These equations can be written in a more compact vector equation:

$$\frac{\partial}{\partial t}\int_{\Omega} \mathbf{u} \, da + \int_{\partial\Omega} \mathbf{F}(\mathbf{u})\cdot\mathbf{n} \, dl = 0 \qquad (5.5)$$

with

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho\mathbf{V} \\ E \end{pmatrix}, \quad \mathbf{F}(\mathbf{u})\cdot\mathbf{n} = \begin{pmatrix} \rho(\mathbf{V}\cdot\mathbf{n}) \\ \rho\mathbf{V}(\mathbf{V}\cdot\mathbf{n}) + p\mathbf{n} \\ (E+p)(\mathbf{V}\cdot\mathbf{n}) \end{pmatrix}$$

In the next section, we show the natural extension of the scalar advection scheme to include (5.5).

5.2 Extension of Scalar Advection Schemes to Systems of Equations

The extension of the scalar advection schemes to the Euler equations requires two rather minor modifications:

(1) *Vector Flux Function.* The scalar flux function is replaced by a vector flux function. In the present work, the mean value linearization due to Roe [47] is used. The form of this vector flux function is identical to the scalar flux function (4.53), i.e.

$$\begin{aligned} \mathbf{h}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) =& \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^+, \mathbf{n}) + \mathbf{f}(\mathbf{u}^-, \mathbf{n})\right) \\ & - \frac{1}{2}|A(\tilde{\mathbf{u}})|\left(\mathbf{u}^+ - \mathbf{u}^-\right) \end{aligned} \qquad (5.6)$$

where $\mathbf{f}(\mathbf{u},\mathbf{n}) \equiv \mathbf{F}(\mathbf{u}) \cdot \mathbf{n}$, and $A = d\mathbf{f}/d\mathbf{u}$ is the flux Jacobian.

(2) *Componentwise limiting.* The solution variables are reconstructed componentwise. In principle, any set of variables can be used in the reconstruction (primitive variables, entropy variables, etc.). Note that conservation of the mean can make certain variable combinations more difficult to implement than others because of the nonlinearities that may be introduced. The simplest choice is obviously the conserved variables themselves. When conservation of the mean is not important (steady-state calculations), we prefer the use of primitive variables in the reconstruction step.

The resulting scheme for the Euler equations has the same shock resolving characteristics as the scalar scheme. Figures 5.1a-b show a simple Steiner triangulation and the resulting solution obtained with a linear reconstruction scheme for transonic Euler flow ($M_\infty = .80, \alpha = 1.25°$) over a NACA 0012 airfoil section.

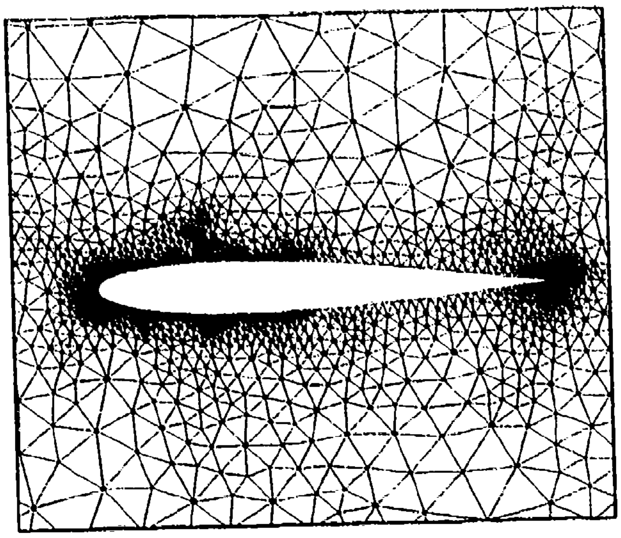


Figure 5.1a Initial triangulation of airfoil, 3155 vertices.

Even though the grid is very coarse with only 3155 vertices, the upper surface shock is captured cleanly with a profile that extends over two cells of the mesh.

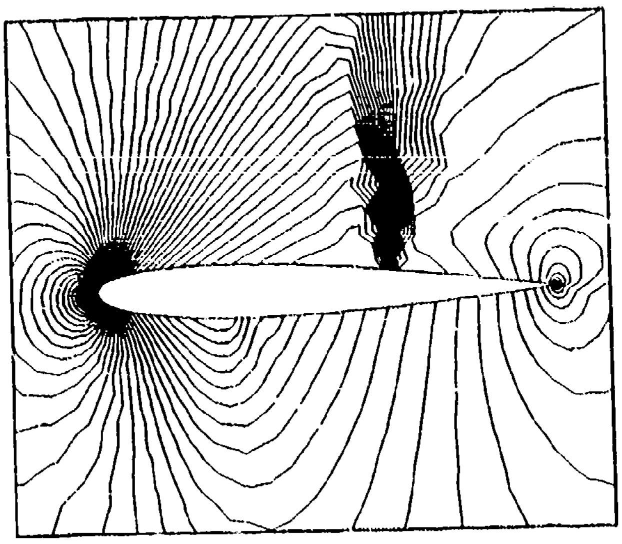


Figure 5.1b Mach number contours on initial triangulation, $M_\infty = .80, \alpha = 1.25°$.

Clearly, the power of the unstructured grid method is the ability to locally adapt the mesh to resolve flow features. Figures 5.2a-b show an adaptively refined mesh and solution for the same flow. The mesh has been locally refined based on *a posteriori* error estimates. These estimates were obtained by performing k-exact reconstruction in each control volume using linear and quadratic functions. A complete discussion of error estimation and solution adaption will be given by Professor Johnson in these notes. The paper by Warren *et al* [29] also provides some interesting insights into the area of mesh adaptation for flows containing discontinuities.

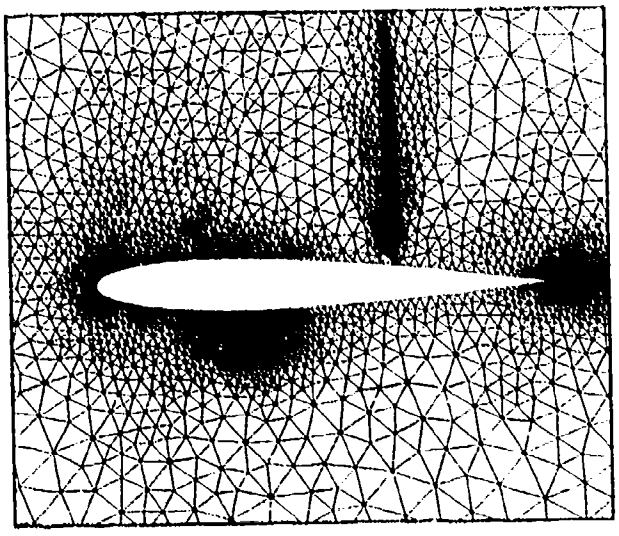


Figure 5.2a Solution adaptive triangulation of airfoil, 6917 vertices.

The flow features in figure 5.2b are clearly defined with a weak lower surface shock now visible. Figure 5.3 shows the surface pressure coefficient distribution on the airfoil. The discontinuities are crisply captured by the scheme.

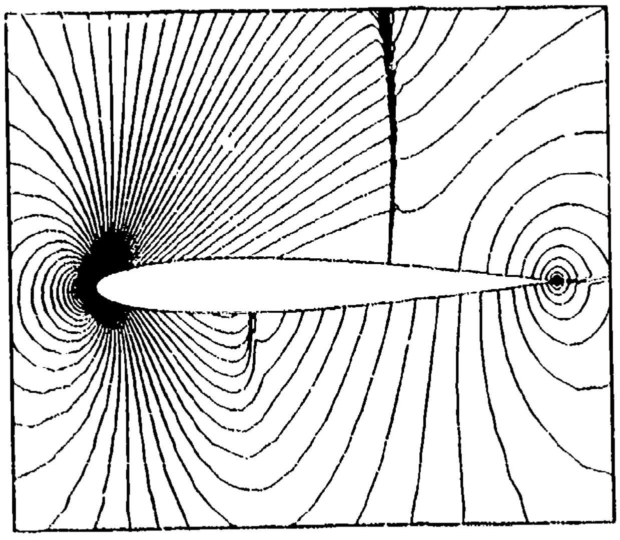


Figure 5.2b Mach number contours on adapted airfoil.

The other major advantage of unstructured grids is the ability to automatically mesh complex geometries. The next example shown in figure 5.4ab is a Steiner triangulation and solution about a multi-component airfoil.

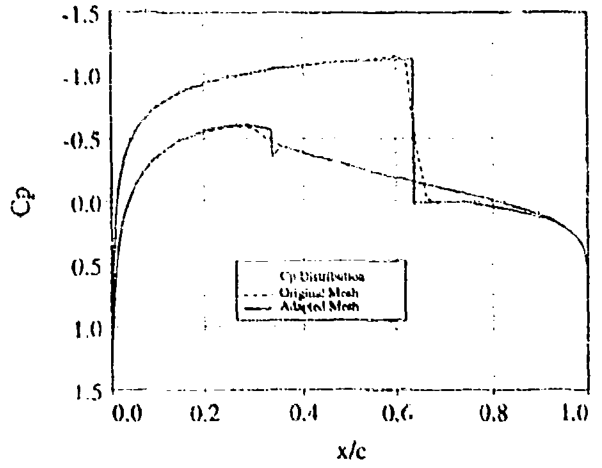


Figure 5.3 Comparison of C_p distributions on initial and adapted meshes.

Using the incremental Steiner algorithm discussed previously, the grid can constructed from curve data in about ten minutes time on a standard engineering workstation using less than a minute of actual CPU time.

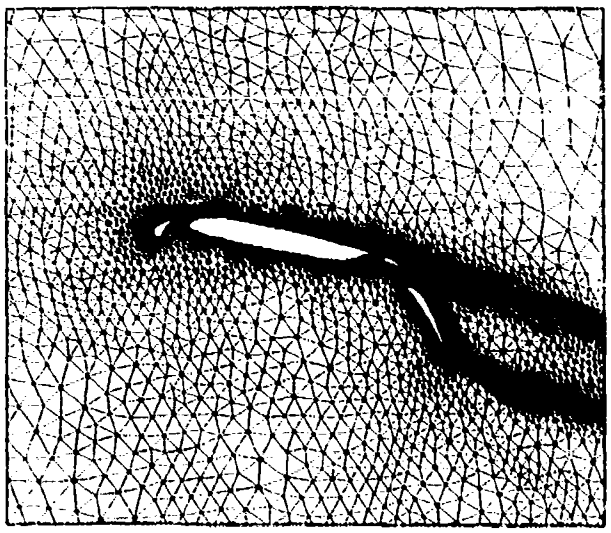


Figure 5.4a Steiner Grid about multi-component airfoil.

The flow calculation shown in figure 5.4b was performed on a CRAY supercomputer taking just a few minutes of CPU time using a linear reconstruction scheme with implicit time advancement. Details of the implicit scheme are given in the next section.

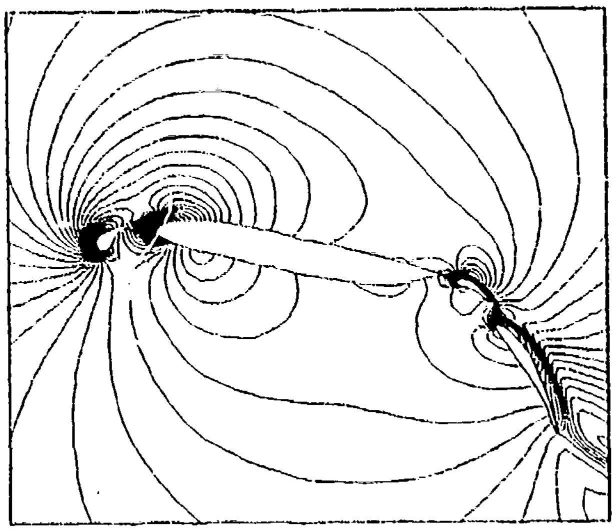


Figure 5.4b Mach number contours about multi-component airfoil, $M_\infty = .2, \alpha = 0°$.

We previously mentioned the importance of using accurate flux quadrature formulas. In fact, for k-exact reconstruction, we suggest N point Gauss quadratures with $N \geq (k+1)/2$. In Figs. 5.5a-b this importance is illustrated by plotting density contours for a numerical calculation of the Ringleb flow (previously described) using quadratic reconstruction $k = 2$. Our formula suggests that two point quadratures should be used in this case.

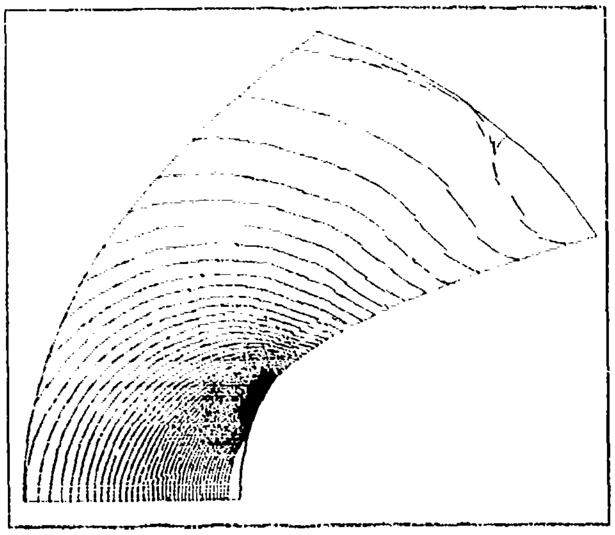


Figure 5.5a Ringleb flow density contours using quadratic reconstruction and one-point Gauss quadrature ($k = 2, N = 1$).

Figure 5.5a shows contours for a calculation using one-point Gauss quadrature and Fig. 5.5b shows contours for a calculation using two-point quadratures. The improvement in Fig. 5.5b is dramatic. Increasing the number of quadrature points to three leaves the solution unchanged.

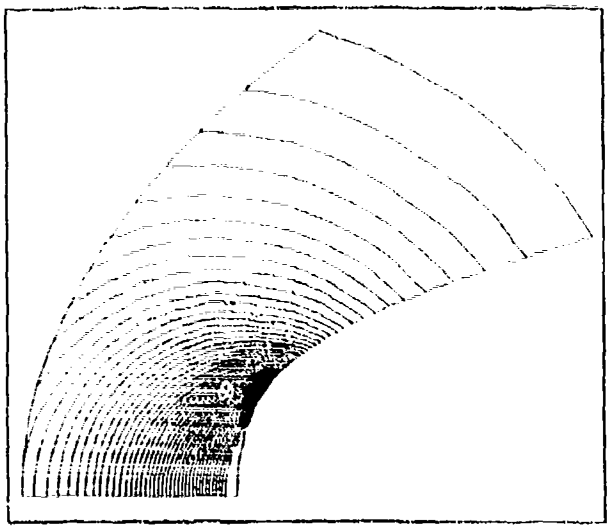


Figure 5.5b Ringleb flow density contours using quadratic reconstruction and two-point Gauss quadrature ($k = 2, N = 2$).

The algorithms outlined in section 4 have been extended to include the Euler equations in three dimensions. In reference [43], we showed the natural extension of the edge data structure in the development of an Euler equation solver on tetrahedral meshes. One of the calculations presented in this paper simulated Euler flow about the ONERA M6 wing. The tetrahedral mesh used for the calculations was a subdivided 151x17x33 hexahedral C-type mesh with spherical wing tip cap. The resulting tetrahedral mesh contained 496,350 tetrahedra, 105,177 vertices, 11,690 boundary vertices, and 23,376 boundary faces. Figure 5.6 shows a closeup of the surface mesh near the outboard tip.

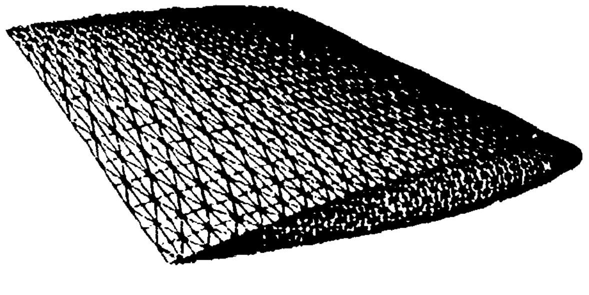


Figure 5.6 Closeup of M6 Wing Surface Mesh Near Tip.

Transonic calculations, $M_\infty = .84, \alpha = 3.06°$, were performed on the CRAY Y-MP computer using the upwind code with both the Green-Gauss and L_2 gradient reconstruction. Figure 5.7 shows surface pressure contours on the wing surface and C_p profiles at several span stations.

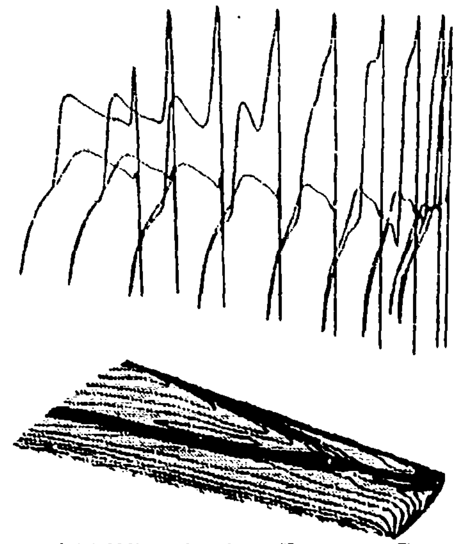


Figure 5.7 M6 Wing Surface Pressure Contours and Spanwise C_p Profiles (M_∞= .84, α=3.06°).

Pressure contours clearly show the lambda type shock pattern on the wing surface. Figures 5.8a-c compare pressure coefficient distributions at three span stations on the wing measured in the experiment, y/b=.44,.65,.95.

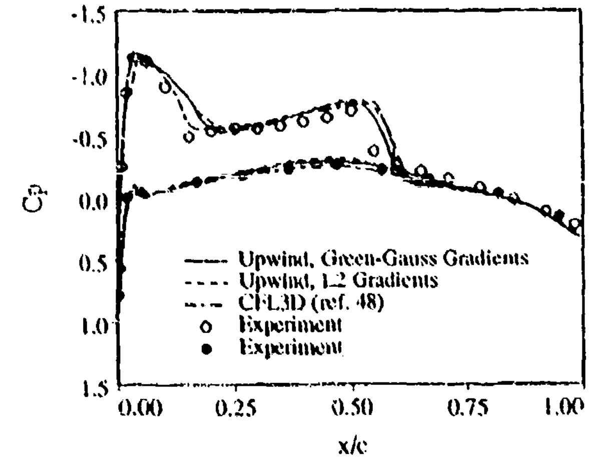


Figure 5.8a M6 Wing Spanwise Pressure Distribution, $y/b = .44$.

Each graph compares the upwind code with Green-Gauss and L_2 gradient calculation with the CFL3D results appearing in [48] and the experimental data [49]. Numerical results on the tetrahedral mesh compare very favorably with the CFL3D structured mesh code. The results for the outboard station appear better for the present code than the CFL3D results. This is largely due to the difference in grid topology and subsequent improved resolution in that area.

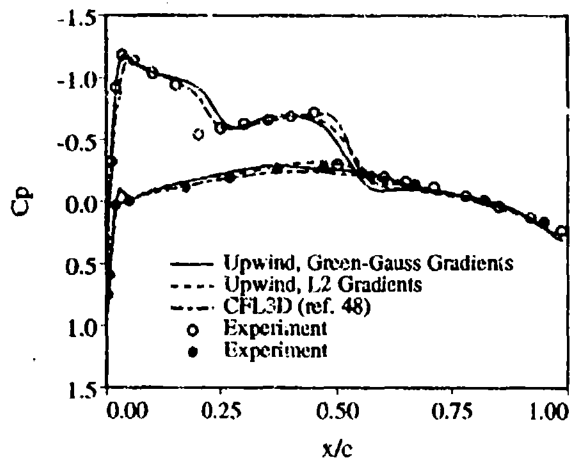


Figure 5.8b M6 Wing Spanwise Pressure Distribution $y/b = .65$.

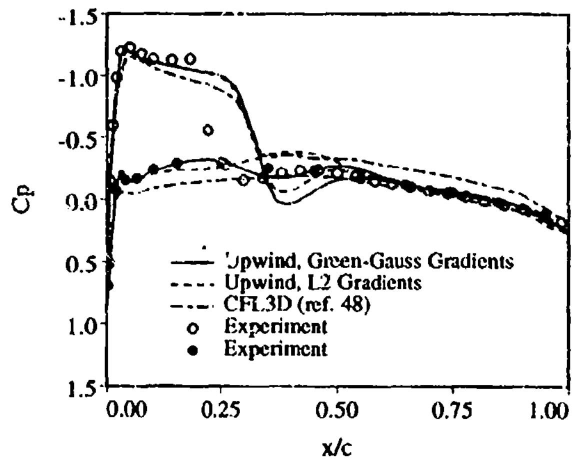


Figure 5.8c M6 Wing Spanwise Pressure Distribution $y/b = .95$.

5.3 Implicit Linearizations

In this section, we consider the task of linearizing the discrete spatial operator for purposes of backward Euler time integration. Defining the solution vector $\mathbf{u} = [\overline{u}_1, \overline{u}_2, \overline{u}_3, ..., \overline{u}_N]^T$, the basic scheme is written as

$$D\mathbf{u}_t = \mathbf{R}(\mathbf{u}) \qquad (5.7)$$

where D is a positive diagonal matrix. Performing a backward Euler time integration, equation (5.7) is rewritten as

$$D(\mathbf{u}^{n+1} - \mathbf{u}^n) = \Delta t\ \mathbf{R}(\mathbf{u}^{n+1}). \qquad (5.8)$$

where n denotes the iteration (time step) level. Linearizing the right-hand-side (RHS) of (5.8) in time produces the following form:

$$D(\mathbf{u}^{n+1} - \mathbf{u}^n) = \Delta t\left(\mathbf{R}(\mathbf{u}^n) + \frac{d\mathbf{R}^n}{d\mathbf{u}}(\mathbf{u}^{n+1} - \mathbf{u}^n)\right)$$

By rearranging terms, we can arrive at the so called "delta" form of the backward Euler scheme

$$\left[D - \Delta t\ \frac{d\mathbf{R}^n}{d\mathbf{u}}\right](\mathbf{u}^{n+1} - \mathbf{u}^n) = +\Delta t\ \mathbf{R}(\mathbf{u}^n) \qquad (5.9)$$

Note that for large time steps, the scheme becomes equivalent to Newton's method for finding roots of a nonlinear system of equations. Newton's method is known to be quadratically convergent for isolated roots. Each iteration of the scheme requires the solution of an algebraic system of linear equation. In practice, we use either sparse direct methods as discussed in ref. [45] or preconditioned minimum residual methods. Both of these topics will be addressed by Professor Hughes and other lecturers. The success or failure of these methods hinges heavily on the accuracy of the time linearization. For the schemes discussed in sections 4 and 5, the most difficult task is the linearization of the numerical flux vector with respect to the two solution states. For example, given the flux vector

$$\begin{aligned}\mathbf{h}(\mathbf{u}^+, \mathbf{u}^-, \mathbf{n}) =& \frac{1}{2}\left(\mathbf{f}(\mathbf{u}^+, \mathbf{n}) + \mathbf{f}(\mathbf{u}^-, \mathbf{n})\right) \\ &- \frac{1}{2}|A(\tilde{\mathbf{u}})|\left(\mathbf{u}^+ - \mathbf{u}^-\right)\end{aligned} \qquad (5.10)$$

we require the Jacobian terms $\frac{d\mathbf{h}}{d\mathbf{u}^+}$ and $\frac{d\mathbf{h}}{d\mathbf{u}^-}$. In reference [50], we derived the exact Jacobian linearization of Roe's flux function. In this same paper, approximate linearizations of (5.10) were investigated. The linearization of (5.10) is straightforward, except for terms which arise from differentiation of $|A(\tilde{\mathbf{u}})|$. A simple approximation is to neglect these terms in the linearization process. This produces the following approximate linearizations:

$$\frac{d\mathbf{h}}{d\mathbf{u}^+} = \frac{1}{2}(A(\mathbf{u}^+) - |A(\tilde{\mathbf{u}})|) \quad \text{(Approx 1)}$$

$$\frac{d\mathbf{h}}{d\mathbf{u}^-} = \frac{1}{2}(A(\mathbf{u}^-) + |A(\tilde{\mathbf{u}})|) \quad \text{(Approx 1)}$$

It is not difficult that to prove that the error associated with this approximation is $O(\|\mathbf{u}^+ - \mathbf{u}^-\|)$ which makes the linearization attractive for the implicit calculation of smooth flows. Near discontinuities, this term becomes $O(1)$ which can slow the convergence of the scheme considerably. One important attribute of this approximation is that it retains *time conservation* of the scheme. This amounts to a telescoping property of fluxes in time. For time accurate problems involving moving discontinuities, this property is essential to obtain correct shock speeds. Another approximate form considered in [50] uses the following simple approximation

$$\frac{dh}{du^+} = A(\tilde{u})^- \quad \text{(Approx 2)}$$

$$\frac{dh}{du^-} = A(\tilde{u})^+ \quad \text{(Approx 2)}$$

This linearization also differs from the exact linearization by terms $O(\|u^+ - u^-\|)$. One important feature of this linearization is that it produces a LHS operator for the first order upwind scheme which is (block) diagonally dominant. For those solution methods or preconditioning methods based on classical relaxation schemes, this property establishes convergence of the relaxation method. Scalar equation analysis also indicates that when this linearization is used with backward Euler time integration and first order upwind space discretization, the resulting scheme is monotone for all time step size. Unfortunately, this linearization violates time conservation and should not be used for time accurate calculations.

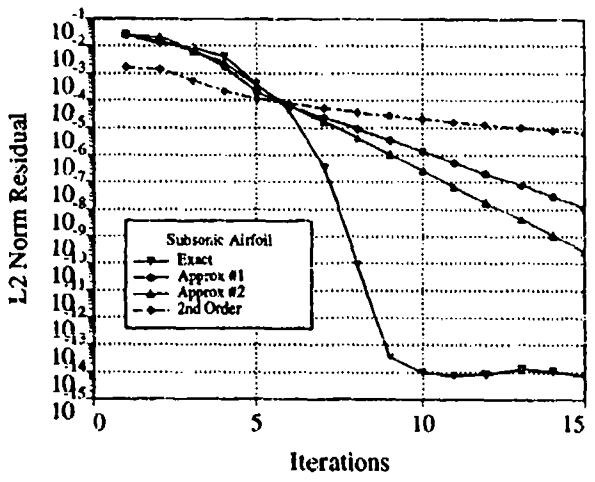


Figure 5.9 Convergence histories for exact and approximate linearizations. Solid lines show convergence histories for calculations carried out using first order upwind RHS in(5.9). Dashed line depicts scheme run with exact linearization of first order scheme on the LHS and second order RHS discretization.

Using the edge data structure, the assembly of the LHS matrix in (5.9) for the first order scheme is very straightforward. The flux associated with each edge $e(v_i, v_j)$ of the mesh is linearized with respect its two arguments, u_i and u_j. This means that the linearization contributes to the formation of the block matrix elements in the i-th row j-th column, i-th row i-th column, j-th row i-th column, and j-th row j-th column positions. This leads to a highly vectorizable (using gather-scatter hardware) algorithm for matrix assembly (and matrix multiplies). For the higher order reconstruction schemes, the usual strategy is to only construct LHS matrix terms associated with the first order upwind scheme while using a higher order RHS operator. The mismatch of operators destroys any hope of quadratic convergence for large time steps. Figure 5.9 graphs the convergence history for a typical calculation using the linearizations discussed above. The flow problem being solved is subsonic flow over a single airfoil. In this case, the flow is smooth and all linearizations should be applicable. In this figure, we see that when the RHS and LHS operators both correspond to the first order upwind scheme and the exact Jacobian linearization is used, quadratic convergence is achieved. The schemes using approximate linearizations do not approach quadratic convergence but are very effective in reducing the initial residual. In reality, most computations are terminated after reducing the residual about four orders of magnitude. For the present example, this would amount to 7 steps using the exact linearization or 8-9 steps using the approximate forms. Using a higher order accurate RHS slows the convergence even further. Nevertheless, a four order reduction in residual is achieved after 30-40 steps.

6.0 Numerical Solution of the Navier-Stokes Equations with Turbulence

6.1 Turbulence Modeling for Unstructured Grids

Simulating the effects of turbulence on unstructured meshes via the compressible Reynolds-averaged Navier-Stokes equations and turbulence modeling is a relatively unexplored topic. In early work by Rostand [51], an algebraic turbulence model was incorporated into an unstructured mesh flow solver. This basic methodology was later refined by Mavriplis [52] for the flow about multi-element airfoil configurations. Both of these implementations utilize locally structured meshes to produce one-dimensional-like boundary-layer profiles from which algebraic models can determine an eddy viscosity coefficient for use in the Reynolds-averaged Navier-Stokes equations. The use of local structured meshes limits the general applicability of the method.

The next level of turbulence modeling involves the solution of one or more auxiliary differential equations. Ideally these differential equations would only require initial data and boundary conditions in the same fashion as the Reynolds-averaged mean equations. The use of turbulence models based on differential equations greatly increases the class of geometries that can be treated "automatically." Unfortunately this does not make the issue of turbulence modeling a "solved" problem since most turbulence models do not perform well across the broad range of flow regimes usually generated by complex geometries. Also keep in mind that most turbulence models for wall-bounded flow require knowledge of distance to the wall for use in "damping functions" which simulate the damping effect of solid walls on turbulence. The distance required in these models is measured in "wall units" which means that physical distance from the wall y is scaled by the local wall shear, density, and viscosity.

$$y^+ = \sqrt{\frac{\tau_{wall}}{\rho_{wall}}}\frac{y}{\nu} \tag{6.1}$$

Scaling by wall quantities is yet another complication but does not create serious implementation difficulties for unstructured meshes as we will demonstrate shortly.

6.2 A One-Equation Turbulence Transport Model

In a recent report with Baldwin [53], we proposed and tested (on structured meshes) a single equation turbulence transport model. In this report, the model was tested on various subsonic and transonic flow problems: flat plates, airfoils, wakes, etc. The model consists of a single scalar advection-diffusion equation with source term for a field variable which is the product of turbulence Reynolds number and kinematic viscosity, $\nu\tilde{R}_T$. This variable is proportional to the eddy viscosity except very near a solid wall.

$$\begin{aligned}\frac{D(\nu\tilde{R}_T)}{Dt} &= (c_{\epsilon_2}f_2 - c_{\epsilon_1})\sqrt{\nu\tilde{R}_T P} \\ &+(\nu + \frac{\nu_t}{\sigma_R})\nabla^2(\nu\tilde{R}_T) - \frac{1}{\sigma_\epsilon}(\nabla\nu_t)\cdot\nabla(\nu\tilde{R}_T)\end{aligned} \tag{6.2}$$

In this equation, P is the production of turbulent kinetic energy and is related to the mean flow velocity rate-of-strain and the kinematic eddy viscosity ν_t. In equation (6.2), the following functions are required:

$$\begin{aligned}\frac{1}{\sigma_\epsilon} &=(c_{\epsilon_2} - c_{\epsilon_1})\sqrt{c_\mu}/\kappa^2 \\ \sigma_R &=\sigma_\epsilon \\ \nu_t &=c_\mu(\nu\tilde{R}_T)D_1 D_2 \\ \mu_t &=\rho\nu_t \\ D_1 &=1 - \exp(-y^+/A^+) \\ D_2 &=1 - \exp(-y^+/A_2^+) \\ P &=\nu_t\left(\frac{\partial U_i}{\partial x_j} + \frac{\partial U_j}{\partial x_i}\right)\frac{\partial U_i}{\partial x_j} - \frac{2}{3}\nu_t\left(\frac{\partial U_k}{\partial x_k}\right)^2\end{aligned}$$

$$\begin{aligned}f_2(y^+) =&\frac{c_{\epsilon_1}}{c_{\epsilon_2}} + (1 - \frac{c_{\epsilon_1}}{c_{\epsilon_2}})(\frac{1}{\kappa y^+} + D_1 D_2)\Bigg(\sqrt{D_1 D_2} \\ &+\frac{y^+}{\sqrt{D_1 D_2}}\Big(\frac{1}{A^+}\exp(-y^+/A^+)\ D_2 \\ &+\frac{1}{A_2^+}\exp(-y^+/A_2^+)\ D_1\Big)\Bigg)\end{aligned}$$

The following constants have been recommended in [53]:

$$\begin{aligned}\kappa &=0.41, & c_{\epsilon_1} &= 1.2, & c_{\epsilon_2} &= 2.0 \\ c_\mu &=0.09, & A^+ &= 26, & A_2^+ &= 10\end{aligned}$$

We also recommend the following boundary conditions for (6.2):

1. **Solid Walls**: Specify $\tilde{R}_T = 0$.
2. **Inflow** ($\mathbf{V}\cdot\mathbf{n} < 0$): Specify $\tilde{R}_T = (\tilde{R}_T)_\infty < 1$.
3. **Outflow** ($\mathbf{V}\cdot\mathbf{n} > 0$): Extrapolate $\tilde{R}_T$ from interior values.

Equation (6.2) depends on distance to solid walls in two ways. First, the damping function f_2 appearing in equation (6.2) depends directly on distance to the wall (in wall units). Secondly, ν_t depends on $\nu\tilde{R}_t$ and damping functions which require distance to the wall.

$$\nu_t = c_\mu D_1(y^+)D_2(y^+)\nu\tilde{R}_t$$

It is important to realize that the damping functions f_2, D_1, and D_2 deviate from unity only when very near a solid wall. For a typical turbulent boundary-layer (see figure 6.1) accurate distance to the wall is only required for mesh points which fall below the logarithmic region of the boundary-layer.

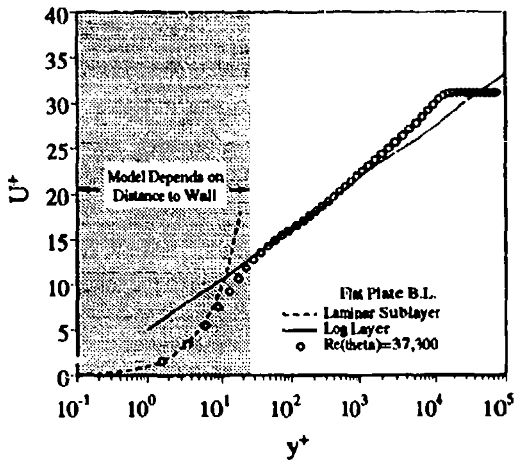


Figure 6.1 Typical flat plate boundary-layer from ref. [53] showing dependence of turbulence model on distance to wall.

The relative insensitivity of distance to the wall means that accurate estimation of distance to the wall is only required for a small number of points that are extremely close to a boundary surface. The remaining points can be assigned any approximate value of physical distance since the damping functions are essentially at their asymptotic values. A general procedure for calculation of distance to the wall in wall units is to precompute and store, for each vertex of the mesh, the minimum distance from the vertex to the closest solid wall (examples are shown later in figures 6.2b and 6.3b). This strategy can only fail if two bodies are in such close proximity that the near wall damping functions never reach their asymptotic values. Realistically speaking, the validity of most turbulence models would be in serious question in this situation anyway. In general, the minimum distance from vertex to boundary edge does not occur at the end points of a boundary edge but rather interior to a boundary edge. For each vertex, information concerning which boundary edge achieves the minimum distance and the weight factor for linear interpolation along the boundary edge must be stored. Data can then be interpolated to the point of minimum distance whenever needed. In the course of solving (6.2), distance to a solid wall in wall units is calculated by retrieving physical distance to the wall and the local wall quantities needed for (6.1) as interpolated along the closest boundary edge.

The numerical calculations presented in this section represent a successful application of the ideas discussed in previous sections. Figures 6.2a and 6.3a show examples of Min-Max triangulations about single and multi-element airfoils. The first geometry consists of a single RAE 2822 airfoil. Navier-Stokes flow is computed about this geometry assuming turbulent flow with the following free-stream conditions: $M_\infty = .725, \alpha = 2.31°, Re = 6.5$ million. Wind tunnel experiments for the RAE 2822 geometry at these test conditions have been performed by Cook, McDonald, and Firmin [54]. The RAE 2822 airfoil mesh shown in figure 6.2a contains approximately 14000 vertices and 41000 edges. The second geometry consists of a two element airfoil configuration with wind tunnel walls. The inflow conditions assume turbulent flow with $M_\infty = .09$ and $Re = 1.8$ million. Details of the geometry and wind tunnel test results can be found in the report by Adair and Horne [55]. The two element mesh shown in figure 6.3a contains approximately 18000 vertices and 55000 edges.

Both meshes were constructed in two steps. The first step was to generate a Delaunay triangulation of the point cloud. As mentioned earlier, the method of Delaunay triangulation can generate poor triangulations for highly stretched point distributions. Both meshes suffered from nearly collapsed triangles with two small interior angles. As a second step, a Min-Max triangulation was constructed by edge swapping the Delaunay triangulation. Edge swapping repaired both triangulations. Both airfoil geometries were calculated assuming turbulent flow using the one-equation turbulence model (6.2). Level sets of the generalized distance function used in the turbulence model are shown in figures 6.2b and 6.3b.

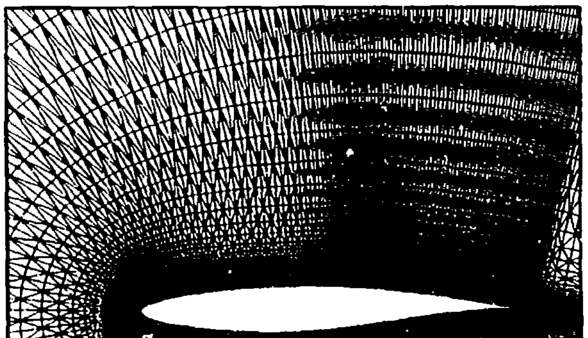


Figure 6.2a Mesh near RAE 2822 airfoil.

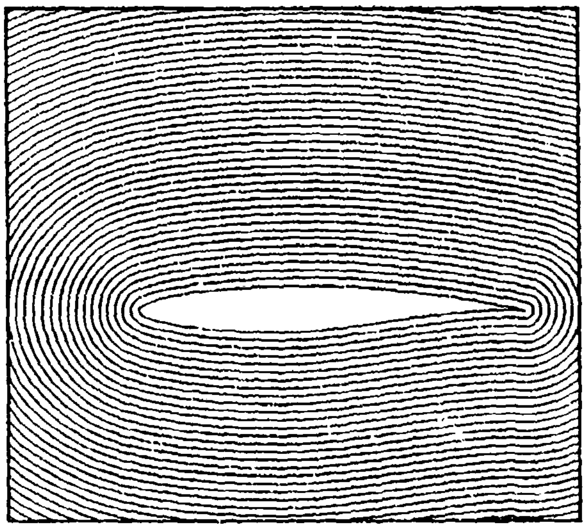


Figure 6.2b Contours of distance function for turbulence model.

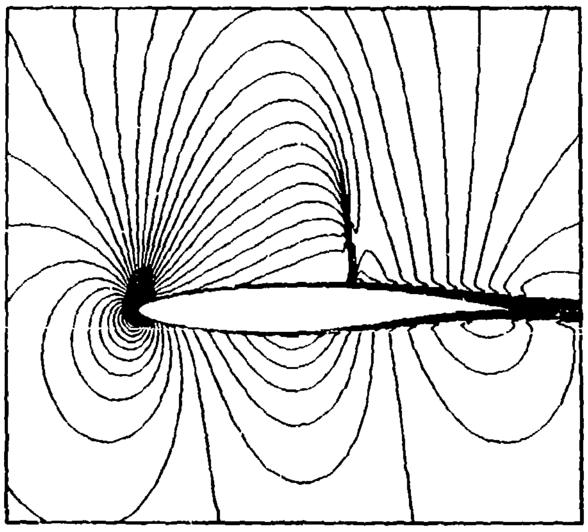


Figure 6.2c Closeup of Mach number contours near airfoil.

Figures 6.2c-d plot Mach number contours and pressure coefficient distributions for the RAE 2822 airfoil. The pressure coefficient distribution compares favorably with the experiment of Cook, McDonald, and Firmin [29]. Leading edge trip strips were used on the experimental model but not simulated in the computations. This may explain the minor differences in the leading edge pressure distribution.

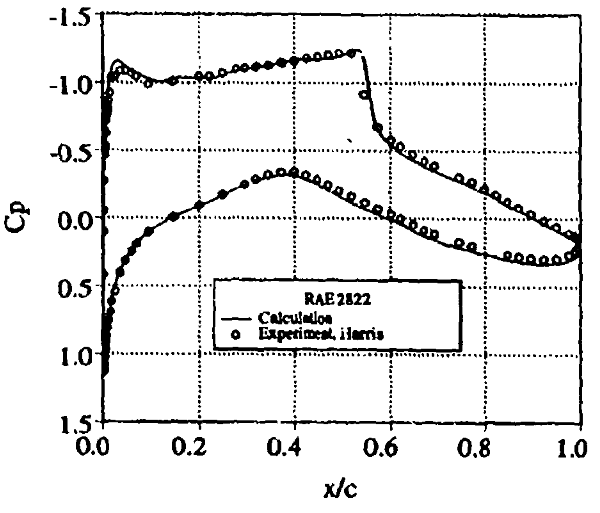


Figure 6.2d Pressure coefficient distribution on airfoil.

Navier-Stokes computations for the two element airfoil configuration as shown in figures 6.3c-d. The effects of wind tunnel walls have been modeled in the computation by assuming an inviscid wall boundary condition. Mach number contours are shown in figure 6.3c. Observe that the contours appear very smooth, even in regions where the mesh becomes very irregular. This is due to the insistance that linear functions be accurately treated in the flow solver reguardless of mesh irregularities.

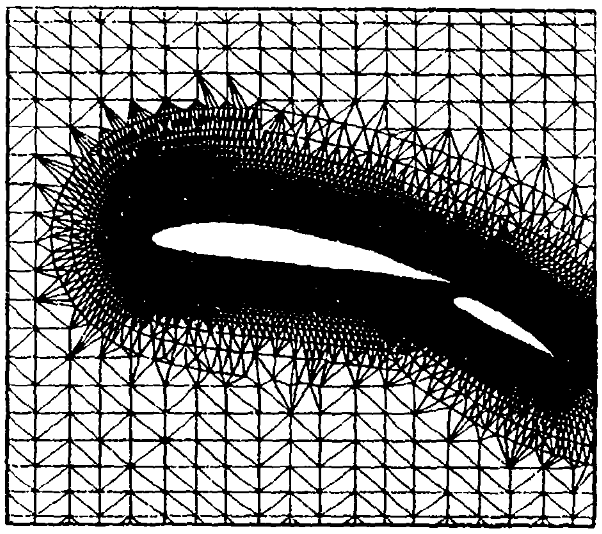


Figure 6.3a Mesh near Multi-element airfoil.

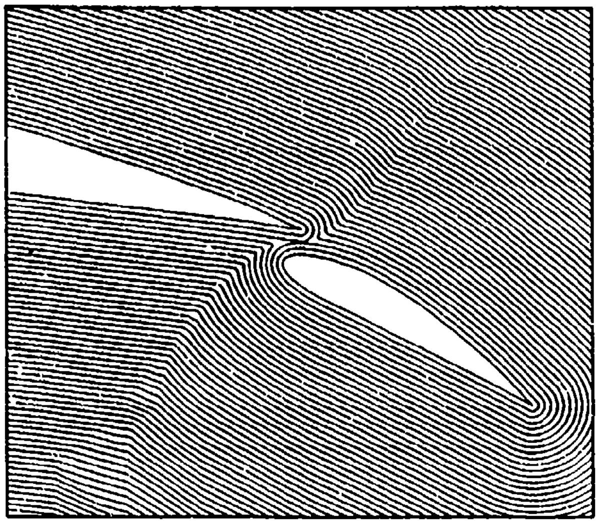


Figure 6.3b Contours of distance function for turbulence model.

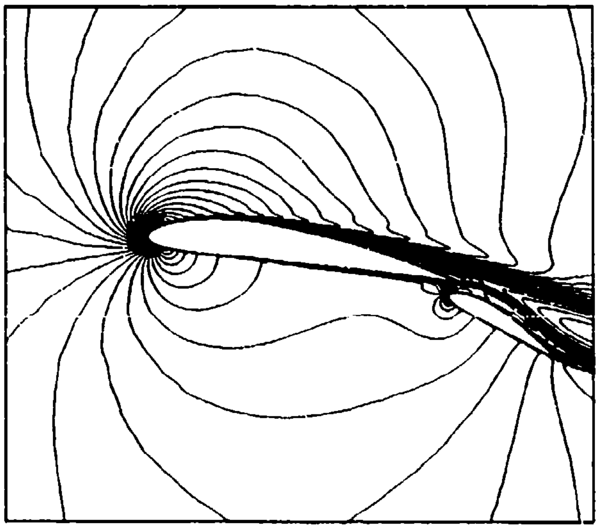


Figure 6.3c Closeup of Mach number contours near airfoil.

Pressure coefficient distribution on the main airfoil and flap are graphed in figure 6.3d. Comparison of calculation and experiment on the main element is very good. The suction peak values of pressure coefficient on the flap element are slightly below the experiment. The experimentors also note a small separation bubble at the trailing edge of the flap which was not found in the computations.

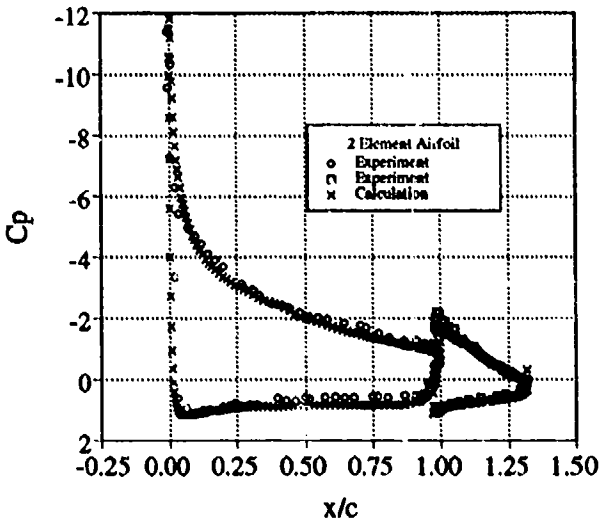


Figure 6.3d Pressure coefficient distribution on airfoil.

References

1. Guibas, L.J., and Stolfi, J.,"Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams",ACM Trans. Graph., Vol. 4, 1985, pp. 74-123.
2. Dobkin, D.P., and Laszlo, M.J.,"Primitives for the Manipulation of Three-dimensional Subdivisions", Algorithmica, Vol. 4, 1989, pp. 3-32.
3. Brisson, E.,"Representing Geometric Structures in *d* Dimensions: Topology and Order", In Proceedings of the 5th ACM Symposium on Computational Geometry., 1989, pp. 218-227.
4. Hammond, S., and Barth, T.J., "An Efficient Massively Parallel Euler Solver for Unstructured Grids", AIAA paper 91-0441, Reno, 1991.
5. Chrobak, M. and Eppstein, D.,"Planar Orientations with Low Out-Degree and Compaction of Adjacency Matrices", Theo. Comp. Sci., Vol. 86, No. 2, 1991, pp.243-266.
6. Rosen, R., "Matrix Band Width Minimization", Proc. ACM Nat. Conf., 1968, pp. 585-595.
7. Cuthill, E., and McKee, J.,"Reducing the Band Width of Sparse Symmetric Matrices", Proc. ACM Nat. Conf., 1969, pp. 157-172.
8. George, J.A,"Computer Implementation of the Finite Element Method", Techical Report No. STAN-CS-71-208, Computer Science Dept., Stanford University, 1971.
9. Venktakrishnan, V., Simon, H.D., Barth, T.J., "A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids", NASA

Ames R.C., Tech. Report RNR-91-024, 1991.

10. Simon, H.D., "Partitioning of Unstructured Problems for Parallel Processing," NASA Ames R.C., Tech. Report RNR-91-008, 1991.
11. Lawson, C. L., "Software for C^1 Surface Interpolation", Mathematical Software III, (Ed., John R. Rice), Academic Press, New York, 1977.
12. Rajan, V. T., "Optimality of the Delaunay Triangulation in $\mathbf{R}^d$", Proceedings of the 7th ACM Symposium on Computational Geometry, 1991, pp. 357-372.
13. Rippa, S., "Minimal Roughness Property of the Delaunay Triangulation", CAGD, Vol. 7, No. 6., 1990, pp-489-497.
14. Bowyer, A., "Computing Dirichlet Tessellations", The Computer Journal, Vol. 24, No. 2, 1981, pp. 162—166.
15. Watson, D. F, "Computing the n-dimensional Delaunay Tessellation with Application to Voronoi Polytopes," The Computer Journal, Vol. 24, No. 2, 1981, pp. 167—171.
16. Baker, T. J. ,"Automatic Mesh Generation for Complex Three-Dimensional Regions Using a Constrained Delaunay Triangulation", Engineering with Computers, Vol. 5, 1989, pp. 161-175.
17. Green, P. J. and Sibson, R., "Computing the Dirichlet Tesselation in the Plane", The Computer Journal, Vol. 21, No. 2, 1977, pp. 168—173.
18. Tanemura, M., Ogawa, T., and Ogita, N.,"A New Algorithm for Three-Dimensional Voronoi Tesselation", J. Comput. Phys., Vol. 51, 1983, pp. 191-207.
19. Merriam, M. L.,"An Efficient Advancing Front Algorithm for Delaunay Triangulation", AIAA paper 91-0792, Reno, NV, 1991.
20. Klee, V., "On the Complexity of d-dimensional Voronoi diagrams", Archiv der Mathematik, Vol. 34, 1980.
21. Lawson, C. L., "Properties of n-dimensional Triangulations" CAGD, Vol. 3, April 1986, pp. 231-246.
22. Babuška, I., and Aziz, A. K., "On the Angle Condition in the Finite Element Method", SIAM J. Numer. Anal., Vol. 13, No. 2, 1976.
23. Edelsbrunner, H., Tan, T.S, and Waupotitsch, R.,"An $O(n^2 \log n)$ Time Algorithm for the MinMax Angle Triangulation," Proceedings of the 6th ACM Symposium on Computational Geometry, 1990, pp. 44-52.
24. Wiltberger, N. L., Personal Communication, NASA Ames Research Center, M.S. 258-1, Moffett Field, CA, 1991.
25. Gilbert, P.N., "New Results on Planar Triangulations", Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, July 1979.
26. Nira, D., Levin, D., Rippa, S.,"Data Dependent Triangulations for Piecewise Linear Interpolation", J. Numer. Anal., Vol. 10, No. 1, 1990, pp. 137-154.
27. Nira, D., Levin, D., Rippa, S.,"Algorithms for the Construction of Data Dependent Triangulations", **Algorithms for Approximation, II**, Chapman, and Hall, London, 1990, [illegible] '98.
28. [illegible]olmes, G. and Snyder, D., "The Genera[illegible] [illegible]f Unstructured Triangular Meshes using [illegible] launay Triangulation," in *Numerical Gr[illegible] Generation in CFD*, pp. 643-652, Pineridge Press, 1988.
29. Warren, G., Anderson, W.K., Thomas, J.L., and Krist, S.L.,"Grid Convergence for Adaptive Methods,", AIAA paper 91-1592-CP, Honolulu, Hawaii, June 24-27,1991.
30. Anderson, W.K.,"A Grid Generation and Flow Solution Method for the Euler Equations on Unstructured grids," NASA Langley Research Center, USA, unpublished manuscript, 1992.
31. Joe, B.,"Three-Dimensional Delaunay Triangulations From Local Transformations", SIAM J. Sci. Stat. Comput., Vol. 10, 1989, pp. 718-741.
32. Joe, B.,"Construction of Three-Dimensional Delaunay Triangulations From Local Transformations", CAGD, Vol. 8, 1991, pp. 123-142.
33. Gandhi, A. S., and Barth T. J.,"3-D Unstructured Grid Generation and Refinement Usinge 'Edge-Swapping' ", NASA TM in preparation, 1992.
34. Godunov, S. K., "A Finite Difference Method for the Numerical Computation of Discontinuous Solutions of the Equations of Fluid Dynamics", Mat. Sb., Vol. 47, 1959.
35. Van Leer, B., "Towards the Ultimate Conservative Difference Schemes V. A Second Order Sequel to Godunov's Method", J. Comp. Phys., Vol. 32, 1979.
36. Colella, P., Woodward, P., "The Piecewise Parabolic Method for Gas-Dynamical Simulations", J. Comp. Phys., Vol. 54, 1984.
37. Woodward, P., Colella, P., "The Numerical Simulation of Two-Dimensioal Fluid Flow with Strong Shocks" J. Comp. Phys., Vol. 54, 1984.
38. Harten, A. , Osher, S., "Uniformly High-

Order Accurate Non-oscillatory Schemes, I.," MRC Technical Summary Report 2823, 1985.

39. Harten, A., Engquist, B., Osher, S., Chakravarthy, "Uniformly High Order Accurate Essentially Non - Oscillatory Schemes III, ICASE report 86-22, 1986.

40. Barth, T. J., and Jespersen, D. C., "The Design and Application of Upwind Scehemes on Unstructured Meshes", AIAA-89-0366, Jan. 9-12, 1989.

41. Barth, T. J., and Frederickson, P. O., "Higher Order Solution of the Euler Equations on Unstructured Grids Using Quadratic Reconstruction", AIAA-90-0013, Jan. 8-11, 1990.

42. Chiocchia, G., " Exact Solutions to Transonic and Supersonic Flows", AGARD Advisory Report AR-211,1985.

43. Barth, T. J.,"A Three-Dimensional Upwind Euler Solver of Unstructured Meshes," AIAA Paper 91-1548, Honolulu, Hawaii, 1991.

44. Struijs, R., Vankeirsblick, and Deconinck, H., "An Adaptive Grid Polygonal Finite Volume Method for the Compressible Flow Equations," AIAA-89-1959-CP, 1989.

45. Barth, T.J.,"Numerical Aspects of Computing Viscous High Reynolds Number Flows on Unstructured Meshes", AIAA paper 91-0721, January, 1991.

46. Ciarlet, P.G., Raviart, P.-A., "Maximum Principle and Uniform Convergence for the Finite Element Method", Comp. Meth. in Appl. Mech. and Eng., Vol. 2., 1973, pp. 17-31.

47. Roe, P.L.,"Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes", J. Comput. Phys., Vol 43, 1981.

48. Thomas, J.L., van Leer, B., and Walters, R.W., "Implicit Flux-Split Schemes for the Euler Equations," AIAA paper 85-1680, 1985.

49. Schmitt, V., and Charpin, F., "Pressure Distributions on the ONERA M6-Wing at Transonic Mach Numbers," in "Experimental Data Base for Computer Program Assessment," AGARD AR-138, 1979.

50. Barth, T.J.,"Analysis of Implicit Local Linearization Techniques for Upwind and TVD Schemes," AIAA Paper 87-0595, 1987.

51. Rostand, P., "Algebraic Turbulence Models for the Computation of Two-dimensional High Speed Flows Using Unstructured Grids," ICASE Report 88-63, 1988.

52. Mavriplis, D., "Adaptive Mesh Generation for Viscous Flows Using Delaunay Triangulation," ICASE Report No. 88-47,1988.

53. Baldwin, B.S., and Barth, T.J.,"A One-Eqn Turbulence Transport Model for High Reynolds Number Wall-Bounded Flows," NASA TM-102847, August 1990.

54. Cook, P.H., McDonald M.A., Firmin, M.C.P., "AEROFOIL RAE 2822 Pressure Distributions, and Boundary Layer and Wake Measurements," AGARD Advisory Report No. 139, 1979.

55. Adair, D., and Horne, W.C., "Characteristics of Merging Shear Layers and Turbulent Wakes of a Multi-element Airfoil," NASA TM 100053, Feb. 1988.

Higher Order Upwind Finite Volume Schemes with ENO-properties for General Unstructured Meshes*

P. Vankeirsbilck†
H. Deconinck‡
CFD Group - von Karman Institute for Fluid Dynamics
Waterloosesteenweg 72
B-1640 Sint-Genesius-Rode
Belgium

1 SUMMARY

This contribution presents high resolution upwind finite volume schemes based on a polynomial reconstruction of the unknowns on unstructured polygonal cells. The schemes are Essentially Non-Oscillatory (ENO) through the use of an adaptive stencil selection. A complete analysis and comparison given in this text shows that the order of the space accuracy of the schemes is at least equal to the degree of the reconstruction polynomial. Numerical results are shown for a nonlinear hyperbolic conservation equation, confirming the ENO shock capturing and higher order accuracy on highly irregular grids. More realistic Euler calculations will demonstrate the ability of the concepts that are outlined theoretically.

2 INTRODUCTION

The aim of the present work is to investigate adaptive unstructured grid Finite Volume methods for the computation of 2D viscous compressible flows. A general approach is followed, based on unstructured polygonal finite volumes with an arbitrary number of edges and with a cell centered definition of the unknowns, ref. [1, 2]. This includes as a subset the classical structured grids based on quadrilaterals as well as the unstructured grids based on triangles. All types of meshes can be encapsulated in just one and the same datastructure which was cast into an object-oriented database written in C++ in view of higher order schemes. This text, however, will not treat convection-diffusion problems but only purely hyperbolic problems of the following form:

$$\frac{\partial q}{\partial t} + \vec{\nabla}.\vec{F}(q,\vec{r}) = S(q,\vec{r}) \quad (1)$$

in which q is a set of solution variables, $\vec{r}$ is any location in the domain Ω where the equation is to be solved and S is a source term. The vector $\vec{F} = (F, G)$ contains the fluxes of the convection quantity in the x- and y-direction.
Eq. (1) can be cast in integral form using Green's theorem as:

$$\frac{\partial}{\partial t}\iint_\Omega q.d\Omega + \oint_\Gamma \vec{F}(q,\vec{r}).\vec{n}.d\Gamma = \iint_\Omega S(q,\vec{r}).d\Omega \quad (2)$$

or, after division by Ω:

$$< q_t >^\Omega + \frac{1}{\Omega}\oint_\Gamma \vec{F}.\vec{n}.d\Gamma = < S(q,\vec{r}) >^\Omega \quad (3)$$

in which $\vec{n}$ is the outward pointing unit normal at the boundary Γ of Ω and $< . >^\Omega$ is an averaging operator over the domain Ω. The flux $\vec{F}.\vec{n}$ will be denoted by H. In the case of the Euler equations, q, H and S are given as follows:

$$\begin{aligned} q &= (\rho, \rho u, \rho v, \rho e)^T \\ H(q,\vec{r}) &= \begin{pmatrix} \rho u_n \\ \rho u_n.u + p.n_x \\ \rho u_n.v + p.n_y \\ \rho u_n.(e + p/\rho) \end{pmatrix} \\ S(q,\vec{r}) &\equiv 0 \end{aligned} \quad (4)$$

with $u_n = \vec{u}.\vec{n}$ the velocity in the normal direction, $\vec{n} = (n_x, n_y)$, ρ the density, u and v the velocity components, e the total energy per unit mass, p the static pressure. The next relations exist between e, p and the temperature T:

$$\begin{aligned} p &= \rho.(\gamma-1).\left(e - \frac{u^2+v^2}{2}\right) \\ p &= \rho.R.T \\ c^2 &= \gamma.R.T \end{aligned} \quad (5)$$

with R the universal constant for gases, γ the ratio of the specific heats at constant pressure respectively constant temperature and c the speed of sound.

In the next section, a description is given of a more classic compressible Euler solver for adaptive unstructured grids. These solvers are based on either a constant or a linear representation of the solution in each volume, see [3] and therefore, they can only achieve a space accuracy which is in between order one and order two. The solution representation is discontinuous across cell interfaces and the values on both sides of each cell interface are used as the initial values for a local Riemann problem. In the case of linear solution representations, either limiters or artificial dissipation are introduced in order to maintain monotonicity in the neighbourhood of discontinuities. However, these techniques locally reduce the space accuracy of the scheme to first order. These schemes are said to be Total Variation Diminishing (TVD), see ref. [4, 5, 6]. Extending these solvers towards Navier-Stokes calculations reveals a strong dependency of the obtained solutions on the local mesh quality. This is particularly true when

*Research supported by Avions Marcel Dassault and the Belgian National Fund for Scientific Research (NFWO)
†Member of Doctoral Program at VKI with Belgian Government IWONL grant
‡Professor at the VKI

local grid refinements have been applied in order to have a better spatial resolution.

One of the possibilities to reduce the mesh sensitivity is to increase the spatial accuracy of the schemes used. To achieve this, the solution representation (reconstruction) in each finite volume must be more accurate. Barth [7, 8] proposed in this context to reconstruct the solution in each control volume by a polynomial of higher degree. Based on his ideas, two variants of his higher order polynomial reconstruction algorithm will be presented. Some results produced by these algorithms and a numerical error study will be shown.
A theoretical study concerning the use of these two algorithms in an upwind finite volume solver is carried out in a separate section. This study reveals that schemes using polynomial solution reconstruction are higher order accurate in space regardless the local grid quality. This is crucial for solution adaptive grid procedures as they introduce sudden and considerable changes in the cell size and shape.
Note that other approaches exist to reduce the grid sensitivity in the literature. Essers and Renard [9] for instance, apply a non-conservative correction to the flux based on the numerically computed coefficients in the local truncation error. However, as conservation is no longer guaranteed, discontinuities will not exhibit the correct jump.
Struijs and Deconinck [10, 11] presented their so called fluctuation splitting schemes whereby the total net flux in a triangle is sent to its 3 nodes according to the direction of the convection speeds. Several wave models were set up to find these directions and showed to be very performing in the case of contact discontinuities on irregular meshes.

It is also well known that schemes with a non-constant solution reconstruction are not monotone and therefore not stable. In order to stabilize these higher order schemes, nonlinearities have to be introduced. For the classical TVD schemes, this is achieved by limiting the solution reconstruction, ref. [3, 6, 12]. In the present work, a different path is followed. The support of the reconstruction, i.e. the set of cells used to compute the coefficients of the reconstruction polynomial, is arbitrary; this degree of freedom leaves the room to choose the support in an adaptive way to stabilize the scheme. The nonlinearity lies then in the fact that cells are accepted or rejected from the support after each iteration. Harten et al. [13] proposed an adaptive selection of the stencil which allows to define the polynomial in each control volume such that the derivatives of the reconstruction polynomial are minimal. Their 1D analysis showed that this selection leads to the so called ENO-schemes, i.e. schemes in which the occurring oscillations are only of order k, the degree of the reconstruction polynomial.
Shu and Osher [14] extended this method to two dimensional cartesian meshes using a dimension by dimension reconstruction, while Durlofsky et al. [15] used linear reconstruction with adaptive stencils on triangular meshes. In the present contribution, two generalizations of these support selection algorithms towards unstructured grids consisting of polygonal cells will be discussed in more detail. A first one is based on some marching procedure whereby the marching direction is based on some heuristic criteria such that both the first order gradient of the reconstruction and the variation of that first gradient over the whole of the stencil are minimal. A second selection algorithm suggests a finite number of candidate supports and selects then the one with minimal norm of all derivatives up to an order equal to the order of the reconstruction. Other methods are given in the literature by Harten and Chakravarthy [16] which minimize mathematically all derivatives up to order k.
In a final section, numerical results obtained using schemes with ENO-reconstruction will be presented for nonlinear scalar convection problems as well as for the compressible Euler equations.

3 A CLASSIC TVD SCHEME

3.1 Introduction

Before introducing polynomial reconstruction in section 4 and the schemes that can be built with it (section 5), a classic TVD-scheme for adaptive unstructured grids is discussed in this section. This discussion shows approximately the current status in finite volume solvers, their capabilities and their shortcomings. These shortcomings are to be situated in the fact that finite volume solvers are sensitive to grid irregularities and that their spatial accuracy is reduced to first order because of the use of limiters or artificial dissipation. ENO-schemes as described in sections 5 and 6 try to cure those shortcomings.

Concerning the notation, q denotes always the exact solution of a given analytical problem whereas Q always stands for some discrete solution of a discretized problem.

3.2 Discretization

The eqs. (3) and (4) can be discretized while defining a set of non-overlapping polygonal finite volumes with area Ω_P which cover the computational domain Ω. The discrete unknowns Q_P are associated with the gravity center of the cell. This leads to the next discrete finite volume equations that have to be satisfied for each cell P:

$$\frac{Q_P^{n+1} - Q_P^n}{\Delta t} + \frac{1}{\Omega_P} \sum_R H_{PR}.\Delta s_{PR} = 0 \qquad (6)$$

where the summation extends over all the neighbouring cells R adjacent to cell P, fig. 1.

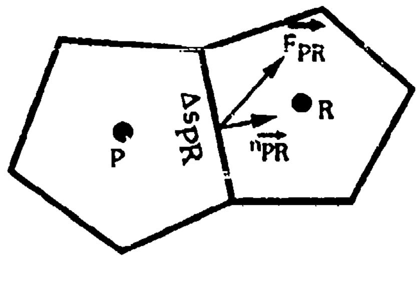


Fig. 1: Flux at the interface between cell P and R.

Here, Δs_{PR} is the length of the edge common to cells P and R. The superscript n stands for the current time level where the the time t is equal to $n.\Delta t$. As the interest here goes mainly to the space accuracy of the scheme, the reader is referred to ref. [3] for more a complete description of the construction of explicit and implicit schemes based on this type of space discretization. H_{PR} is the numerical flux through the edge, defined by an approximate Riemann solver, ref. [17]. In the present approach, Van Leer's flux vector splitting and Roe's flux difference splitting are adopted as approximate Riemann solvers in order to obtain an upwind treatment of the flux. We have for Van Leer [18]:

$$H_{PR} = H^+(Q_P^+) + H^-(Q_R^-) \quad (7)$$

and for Roe's splitter [19]:

$$\begin{aligned} H_{PR} &= \frac{1}{2}\left[H(Q_P^+) + H(Q_R^-)\right] \\ &\quad -\frac{1}{2} \mid C_m(Q_m) \mid \cdot (Q_R^- - Q_P^+) \end{aligned} \quad (8)$$

in which C_m is the Jacobian $\frac{\partial H}{\partial Q}$ taken at the Roe-averaged state $Q_m(Q_P^+, Q_R^-)$. In eqs. (7) and (8), Q_P^+ and Q_R^- indicate any type of extrapolation of the unknowns in cell P resp. R from the gravity centers of these cells to the interface between these two cells. However, in classic finite volume schemes, only constant or linear extrapolation is employed:

$$\begin{aligned} Q_P^+ &= Q_P \\ Q_R^- &= Q_R \end{aligned} \quad (9)$$

in the case of constant solution representation and

$$\begin{aligned} Q_P^+ &= Q_P + (\vec{r}_O - \vec{r}_P).\vec{\nabla}Q_P \\ Q_R^- &= Q_P + (\vec{r}_O - \vec{r}_R).\vec{\nabla}Q_R \end{aligned} \quad (10)$$

for linear solution reconstruction, whereby $\vec{\nabla}Q_P$ and $\vec{\nabla}Q_R$ are assumed to be constant gradient vectors for cell P, resp. cell R. The vector $\vec{r}_O$ points to the midpoint O of the edge between the cells P and R.
On an unstructured mesh, the classic finite difference formulas for approximating the first derivatives of the discrete solution Q in eq. (10) cannot be used. An approximation for the gradient of Q over a given polygon P can be found using the Gauss theorem, fig. 2:

$$\iint_{\Omega_P} \vec{\nabla}.\vec{Q}\, d\Omega = \oint_{\partial\Omega_P} \vec{F}.\vec{n}\, ds \quad (11)$$

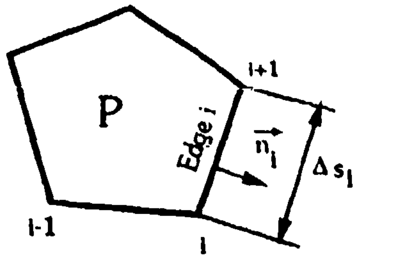


Fig. 2: Gauss theorem.

Indeed, taking $\vec{F} = (Q, 0)$, and assuming $\frac{\partial Q}{\partial x}$ constant over the polygon P, eq. (11) yields:

$$\begin{aligned} \iint_{\Omega_P} \frac{\partial Q}{\partial x} d\Omega &= \oint_{\partial\Omega_P} Q n_x\, ds \\ \Rightarrow \left(\frac{\partial Q}{\partial x}\right)_m &= \frac{1}{\Omega_P} \oint_{\partial\Omega_P} Q n_x\, ds \end{aligned}$$

(the subscript m denoting some point inside the polygon P); in the same way, one finds:

$$\left(\frac{\partial Q}{\partial y}\right)_m = \frac{1}{\Omega_P} \oint_{\partial\Omega_P} Q n_y\, ds \quad (12)$$

and the contour integrals can easily be discretized as a sum over the edges of the control volume:

$$\oint_{\partial\Omega_P} Q n_x\, ds = \sum_i Q_i^* \, (n_x)_i \, \Delta s_i \quad (13)$$

where Q^* is taken as the average of Q between the two nodes defining edge i:

$$Q_i^* = \frac{Q_i + Q_{i+1}}{2} \quad (14)$$

On a mesh with cell-centered storage of the unknowns, the integration contour for estimating the gradient in a given cell P is defined in fig. 3.

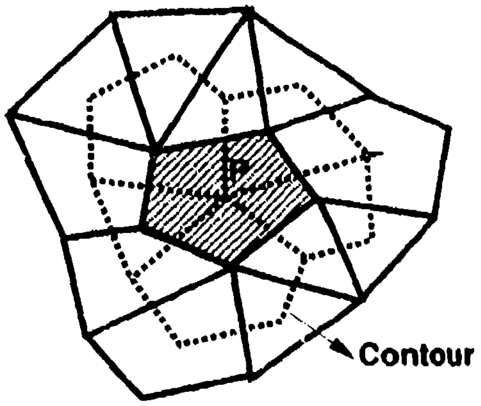


Fig. 3: Integration contour.

Note that the same method is used in a cell adjacent to a boundary of the domain, except that the integration contour is no longer central with respect the cell P, fig. 4. Furthermore, additional unknowns are stored halfway the edges representing the boundary. In the next subsection, it will be shown that these additional unknowns are also useful for treating the boundary conditions more consistently.

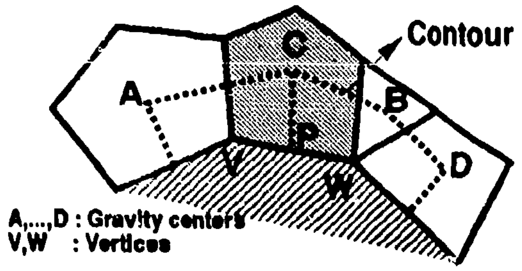


Fig. 4: Contour close to a wall.

Using the gradient computed in the above manner, it is possible to establish a piecewise linear reconstruction of the discrete solution over a cell; i.e., for any point $\vec{r}$ in cell P, we obtain a linear representation $Q_P^{(1)}(\vec{r})$ of the state variables given by eq. (10). If P is taken at the centroid of the cell, it is easy to verify that this linear variation is just a redistribution of the cell averaged data:

$$< Q^{(1)}(\vec{r}) >^P \equiv Q(\vec{r}_P) \equiv Q_P \quad (15)$$

i.e. cell averages and values at the gravity center are exchangeable. Hence, it was indeed justified to use values at the gravity center in eq. (6) when discretizing eq. (3).

It is also important to point out here that the linear reconstruction as presented here will exactly reconstruct the exact solution q of eq. (3) if q is linear. To obtain this, one needs only the values $q_P = q(\vec{r}_{G_P})$ of the solution at the centroid G_P of each cell P. Schemes with this property are called linearity preserving or k-exact, see [8].

Finally, remark also that the scheme given by eq. (6) whereby the gradients in eq. (10) are estimated as described above, is equivalent to the Fromm scheme [20] when applied in one dimension. The Fromm scheme is a member of the family of the so called κ-schemes [21] whereby $\kappa = 0$. The Fromm scheme is upwind biased in the sense that the fluxes in eq. (7) and (8) are taken on one side while the stencil to calculate the extrapolated solution variables is central.

3.3 TVD-Limiters

Schemes using a piecewise linear reconstruction are said to be second order accurate in space on regular grids but are unstable or at least they are highly oscillatory near discontinuities. In order to achieve TVD properties for those schemes, the gradient slopes used for Q_P^+ and Q_R^- in eq. (10) are limited as follows:

$$\begin{aligned} Q_P^+ &= Q_P + \varphi_{PR} \cdot \vec{\nabla} Q_P . (\vec{r}_O - \vec{r}_P) \\ Q_R^- &= Q_R + \varphi_{RP} \cdot \vec{\nabla} Q_R . (\vec{r}_O - \vec{r}_R) \end{aligned} \quad (16)$$

where φ is the limiting function whose value is in most of the cases to be found in the interval $[0,1]$. As an example, van Albada [6] proposed the following form for the limiting function:

$$\varphi = \frac{r^2 + r}{r^2 + 1} \quad (17)$$

where

$$r = sign\,(a.b) . \sqrt{\frac{a^2 + \epsilon}{b^2 + \epsilon}} \approx \frac{a}{b} \quad (18)$$

in which

$$\begin{aligned} a &= (Q_R - Q_P) . \frac{|\vec{r} - \vec{r}_P|}{|\vec{r} - \vec{r}_P| + |\vec{r} - \vec{r}_R|} \\ b &= \vec{\nabla} Q_P . (\vec{r} - \vec{r}_P) \end{aligned} \quad (19)$$

for the computation of Q_P^+ and

$$\begin{aligned} a &= (Q_P - Q_R) . \frac{|\vec{r} - \vec{r}_R|}{|\vec{r} - \vec{r}_P| + |\vec{r} - \vec{r}_R|} \\ b &= \vec{\nabla} Q_R . (\vec{r} - \vec{r}_R) \end{aligned} \quad (20)$$

for the evaluation of Q_R^-. The symbol ϵ denotes a small positive number that avoids division by zero. Other limiters are given in [6, 12].
Note that the limiting can be performed one any type of variables: conservative Q, primitive $\hat{Q}$ or characteristic W variables. For this purpose, the symbol Q should be replaced in the above equations by the proper one: Q, $\hat{Q}$ or W.
The use of equations (16)-(20) guarantees that Q_P^+ and Q_R^- will have a value in the interval $[Q_P, Q_O]$ resp. $[Q_O, Q_R]$, where Q_O is defined as:

$$\begin{aligned} Q_O &= Q_P + (Q_R - Q_P) . \frac{|\vec{r} - \vec{r}_P|}{|\vec{r} - \vec{r}_P| + |\vec{r} - \vec{r}_R|} \\ &= Q_R + (Q_P - Q_R) . \frac{|\vec{r} - \vec{r}_R|}{|\vec{r} - \vec{r}_P| + |\vec{r} - \vec{r}_R|} \end{aligned} \quad (21)$$

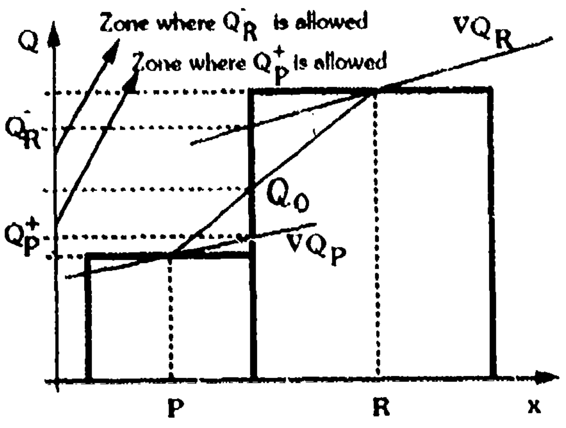


Fig. 5: 1D Representation of the limiting procedure.

In 1D, the value of Q_O corresponds to a linear interpolation at the cell interface between the values of Q_P and Q_R, fig. 5.
By restricting the values of Q_P^+ and Q_R^-, i.e. for values of the function φ close to zero, oscillations are suppressed in the neighbourhood of zones where large gradients occur because the scheme turns locally back to first order.

3.4 Boundary Equations

3.4.1 Flux Equality Method

The flux H_{PR} in eq. (6) through edges at the boundary of the domain are computed by introducing additional unknowns along each boundary edge (fig. 6).

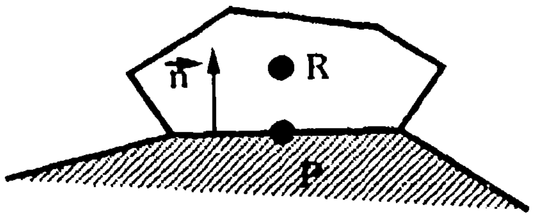


Fig. 6: Boundary edge.

According to [22], the boundary unknowns Q_P are determined by solving the following algebraic consistency equation for the boundary flux H_{PR}, e.g. for van Leer splitting (see eq. (7)):

$$H_{PR} = H^+(Q_P) + H^-(Q_R) = H^*(Q_P) \quad (22)$$

where Q_R is the adjacent cell variable and $H^*(Q)$ a boundary flux function which satisfies the boundary conditions. In other words, the numerical flux H_{PR} must equal the physical flux H^* at the boundaries. E.g. for a solid wall, enforcing zero normal velocity, the fluxes become:

$$H^*(Q_P) = \begin{bmatrix} 0 \\ p(Q_P).n_x \\ p(Q_P).n_y \\ 0 \end{bmatrix}$$

It has been shown by Deconinck et al. [22] that this boundary condition treatment is a characteristic boundary condition treatment in the case of a linear hyperbolic system. For the nonlinear system it is an approximate characteristic boundary condition consistent with the interior approximate Riemann solver.

3.4.2 Flux Balance Method

As proposed by Degrez [23], one updates the variables in th› interior cells adjacent to the walls separately fro .a those being completely inside the mesh.
First, all fluxes through interior edges are computed as usually. In order to update the variables in an interior cell P adjacent to the wall, one uses the flux balance equation (6). However, the fluxes entering the cell through its boundaries that coincide with a physical boundary are yet unknown. Isolating the known fluxes in the flux balance equation (3) gives:

$$< q_t >^P + \frac{1}{\Omega_P} \int_{\partial\Omega_P \setminus \partial\Omega_{P,W}} H.d\Gamma + \int_{\partial\Omega_{P,W}} H.d\Gamma = 0 \tag{23}$$

with $H = \vec{F}.\vec{n}$ and $\partial\Omega_{P,W}$ denoting the edge on the domain boundary. After discretizing, one finds:

$$Q_P^{n+1} - Q_P^n = -\frac{\Delta t}{\Omega_P}\left[\sum_{R\neq W} H_{PR}.\Delta s_{PR} + H_{P,W}.\Delta s_{P,W}\right] \tag{24}$$

Calling the first (known) summation in the right hand side Z, and writing the equations in full whereby $\Delta Q_P = Q_P^{n+1} - Q_P^n$, the next set of equations is found when dealing with a solid wall:

$$\Delta Q_P = Z - \frac{\Delta t}{\Omega_P}\begin{bmatrix} 0 \\ p_W.n_x \\ p_W.n_y \\ 0 \end{bmatrix}.\Delta s_{P,W} \tag{25}$$

in which W denotes the flat boundary ceil bordering the current cell P. One has now a set of 4 equations with 5 unknowns: 4 unknowns in the vector ΔQ_P and the static pressure in the boundary cell W, p_W.
The first equation (mass conservation) can readily be solved, yielding $\Delta\rho_P$ and thus ρ_P^{n+1}, and so can the fourth equation. The velocity vector $\vec{u}_P^{n+1}$ and the wall pressure p_W can be evaluated only if an additional equation can be found. The flux balance method consists now in stating that the value in P (at time step $n+1$), when extrapolated to the boundary cell W using the relevant gradient in the cell (as of time step n), should match the boundary condition. Practically, in case of the flow tangency condition at a solid wall, the additional equation reads

$$\left[\vec{u}_P^{n+1} + \left(\vec{\nabla}\vec{u}_P^n\right).(\vec{r}_W - \vec{r}_P)\right].\vec{n}_{P,W} = 0 \tag{26}$$

This equation allows now to evaluate all variables in the interior cell P at time level $n+1$. However, a few points should be emphasized:

1. when several edges of the interior cell P are boundary edges, extrapolating the variables to each of those boundaries is impossible (it would yield different values to be stored in the cell P). A strategy should be devised to handle such cases, e.g. averaging the extrapolated variables over all boundary edges.

2. when second order space accuracy is used inside the mesh, limiters are used to preserve the monotonicity of the solution, but obviously those limiters should *not* be used when extrapolating variables to the boundaries. The boundary conditions would not be matched exactly.

3. numerical experiments revealed that some underrelaxation of the extrapolations was requested to ensure convergence when second order space accuracy was used inside the mesh, equation (26) becomes then:

$$\left[\vec{u}_P^{n+1} + \omega\left(\vec{\nabla}\vec{u}_P^n\right).(\vec{r}_W - \vec{r}_P)\right].\vec{n}_{P,W} = 0 \tag{27}$$

with ω the relaxation parameter having a value in the interval [0, 1].

3.5 Grid Adaptivity

In order to have sufficient resolution of the calculated solution in regions of high truncation error, locally more and finer cells are needed. In the present work this was done by a simple grid enrichment technique, based on gradients in the solution, ref. [1, 2]. A given cell is refined if the pressure, or alternatively the streamwise entropy gradient both weighted by the cell area satisfy

$$\frac{|\vec{\nabla}p|_P}{RMS}.\frac{\Omega_P}{\Omega_{max}} > C\,,\ \frac{|\vec{u}.\vec{\nabla}s|_P}{RMS}.\frac{\Omega_P}{\Omega_{max}} > C \tag{28}$$

where the gradients are scaled to the RMS value over the field, and C is a user-specified threshold. The cell area Ω_P is normalized with respect to the maximum cell area Ω_{max} over the field. To facilitate the choice of the threshold C, a histogram of the gradient distribution over the mesh cells is calculated as proposed in [24]. Hence, instead of the threshold, the fraction of the total number of cells to be refined is given as an input for the refining. Furthermore, a preliminary view of the refined grid can be obtained by plotting the grid (while connecting the gravity center of all cells satisfying condition (28) to the middle of their edges according to the refinement strategy described below). This view can help to tune the threshold value C in a finer way without having to spend large amounts of computational time in order to set up a new datastructure associated with the refined grid.

A cell is refined by connecting its gravity center with the middle of each edge (fig. 7a and 7b). If the interior cell to be refined lies next to the computational boundary, the boundary cell is split as weil, whereby the newly created boundary cells have the same boundary condition as the original cell. If some of the neighbouring cells are not refined, new vertices have to be created along the refined edges, which increases the number of vertices of this neighbour cell. The conservative variables in the newly created cells are copied from the original cell P.

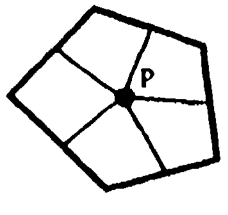


Fig. 7a: Refinement in the interior domain.

The present refinement strategy may lead to non-smooth grids, but has the important merit to limit the remeshing zone to exactly the region targeted by the refinement criterion.

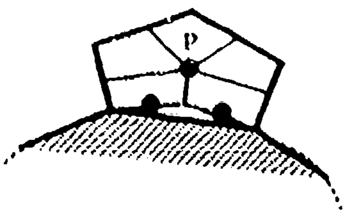


Fig. 7b: Refinement next to a domain boundary.

Anyhow, the non-smoothness of refined grids affects the solution quality dramatically. Especially when several refinements are performed, unacceptable grid irregularities, called "spiders", appear. "Spiders" occur in a cell which has not yet been refined in the previous refinements while some of its neighbours did. Connecting now its gravity center with the middle of each (!) of its edges when refining the cell, leads to the creation of a "spider" (fig. 8) consisting of cells with an odd shape.

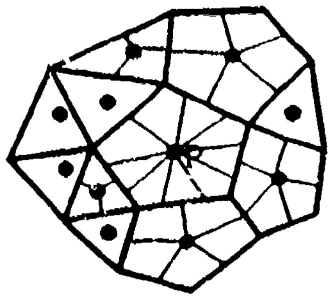


Fig. 8: Generation of a "spider".

This is cured by introducing *level dependent* refinement. This involves assigning a level to each grid cell and each grid vertex. A cell is then refined as if all of its neighbouring cells have the same level, and no "spiders" occur (fig. 9).

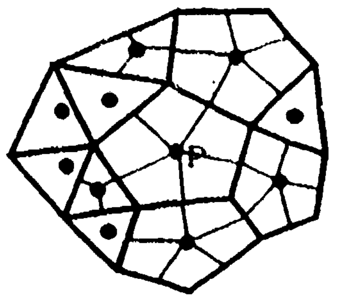


Fig. 9: Level dependent refinement.

The level of all cells and vertices of the starting grid is equal to 1. Fig. 10 shows how the levels are assigned after refinement of a cell. Storage of the grid cell levels is not to be seen as a drawback since these levels will be needed for a future multigrid implementation.
Furthermore, before the refinement starts, some flagged cells for refinement loose their flag again in order to avoid isolated refined cells and spits of refined zones into non-refined zones. The algorithm devised for this turns off the flag of refinement of a cell if at least all but one of its neighbours will not be refined, fig. 11 Obviously, several scans will be needed before the above rule will be fully satisfied.

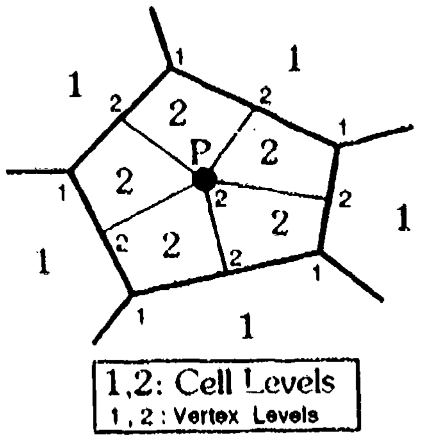


Fig. 10: Assigning levels to the new cells and vertices.

On the other hand, in order to avoid islands and creeks of non-refined cells and a too large difference in cell size between a cell and its neighbours, the following rules are applied after the refinement is finished:

1. A cell is to be refined additionally when all of its neighbours have a higher level (fig. 12a).
2. A cell is to be refined additionally when all except one of its neighbours have a higher level (fig. 12b).
3. A cell is to be refined additionally when the level difference between the cell itself and one of its neighbours exceeds one (fig. 12c). Fig. 12d shows how it is refined.

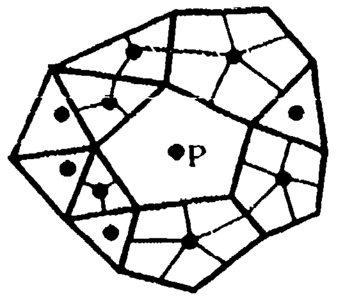


Fig. 12a: All neighbours were refined.

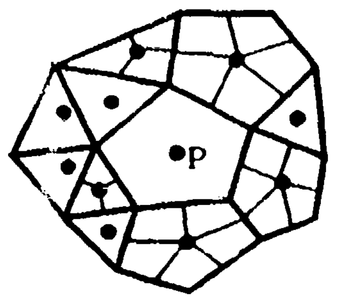


Fig. 12b: All neighbours except one are at a higher level.

Again, several scans of a refined grid are needed to satisfy all the rules, specially when there are deep creeks.

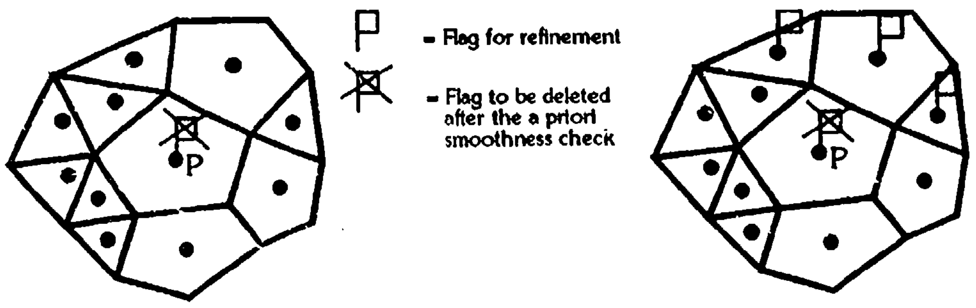


Fig. 11: Rules for avoiding islands and pits of refined cells.

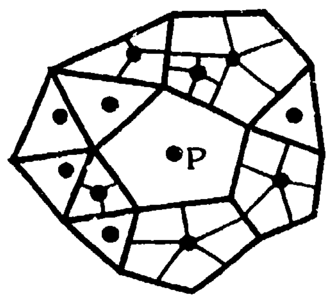


Fig. 12c: The level difference exceeds one.

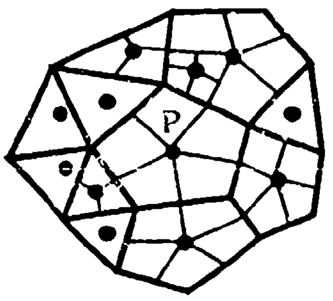


Fig. 12d: Final refinement.

These a priori and a posteriori scans of the mesh to be adapted decrease the precise control on the regions where the grid is to be refined, but the smoothness of the grid is enhanced considerably. Finally, one of the main reasons to use a nested refining as described here instead of a complete (solution adaptive) remeshing is the future implementation of a multigrid acceleration technique.

3.6 Results

This subsection presents the results obtained from a calculation on a 2D supersonic staggered wedge cascade. This test case, for which an analytical solution exists was proposed by Denton [25]; the upstream Mach number is 1.6 and the incidence is 60°. The geometry of the problem is given in fig. 13 while the isentropic Mach number distribution on the profile predicted by the analytical solution is shown in fig. 14. Comparisons will be made with the results of the computation by Holmes [26].
The computation was done using a Van Leer approximate Riemann solver, [18]. In order to perform the time integration a point Gauss-Seidel relaxation technique is used, see ref. [3]. Van Albada's flux limiter was applied

The computation was started on an initial mesh containing 154 cells as shown in fig. 15. Note that the mesh was copied twice in this figure in order to achieve a better visual impression of the cascade. Refining the grid 5 times on basis of the velocity projected entropy gradients in each intermediate solution, the final mesh contained 9333 cells, fig. 16.
As can be seen in fig. 17a, the isentropic Mach number distribution obtained from the final grid hardly exhibits any oscillation. However, note that the dip in the distribution in the lower right corner of the figure is due to the fact that the impinging shock in point A of the wedge is not perfectly cancelled by the expansion wave in the same point. This is presumed to be caused by round-off errors in the specification of the geometry. Comparison of fig. 17a with the result by Holmes [26] as given in fig. 17b reveals clearly the monotonic character of the present scheme. Holmes' result however does not show the dip in the lower right corner of the figure. Both methods have difficulties in predicting the exact value of the isentropic Mach number after the expansion wave in point B of the wedge.
Comparison of the isentropic Mach lines obtained with the two methods shows again that the cancelling of the impinging shock and the expansion shock in point A is not complete with the present method, fig. 18a and 18b.

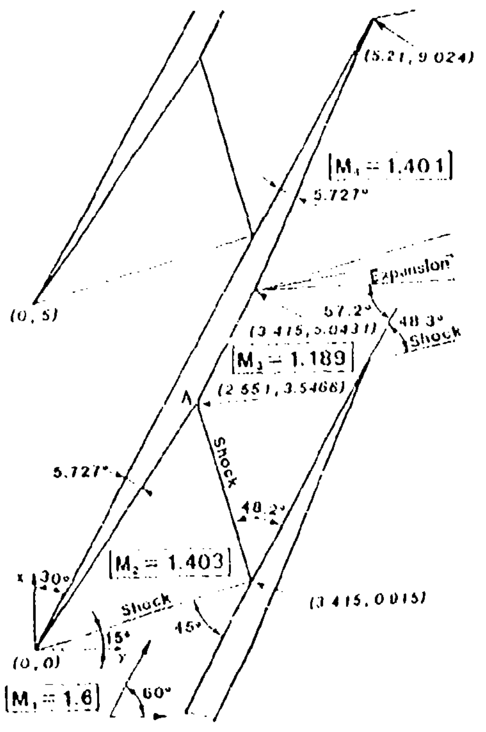


Fig. 13: The Geometry of the Problem.

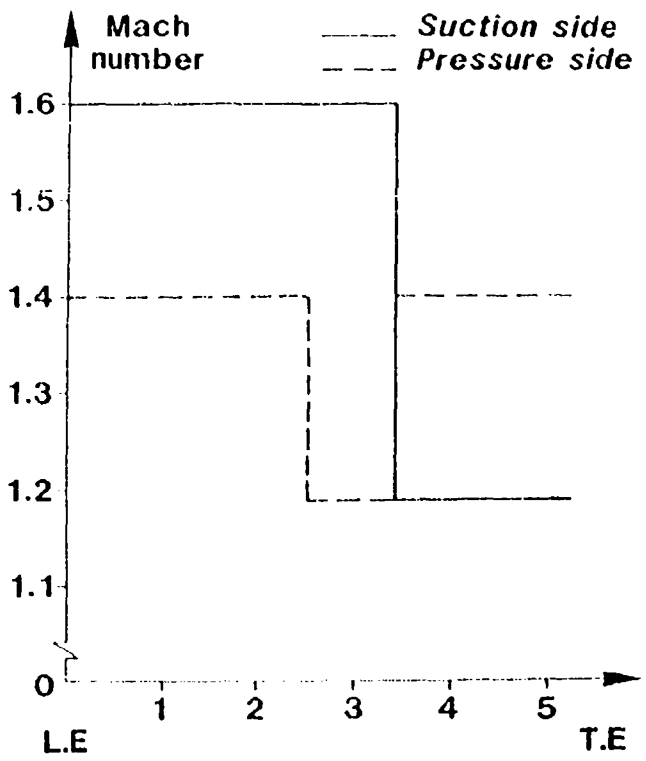


Fig. 14: The Analytical Isentropic Mach Number Distribution

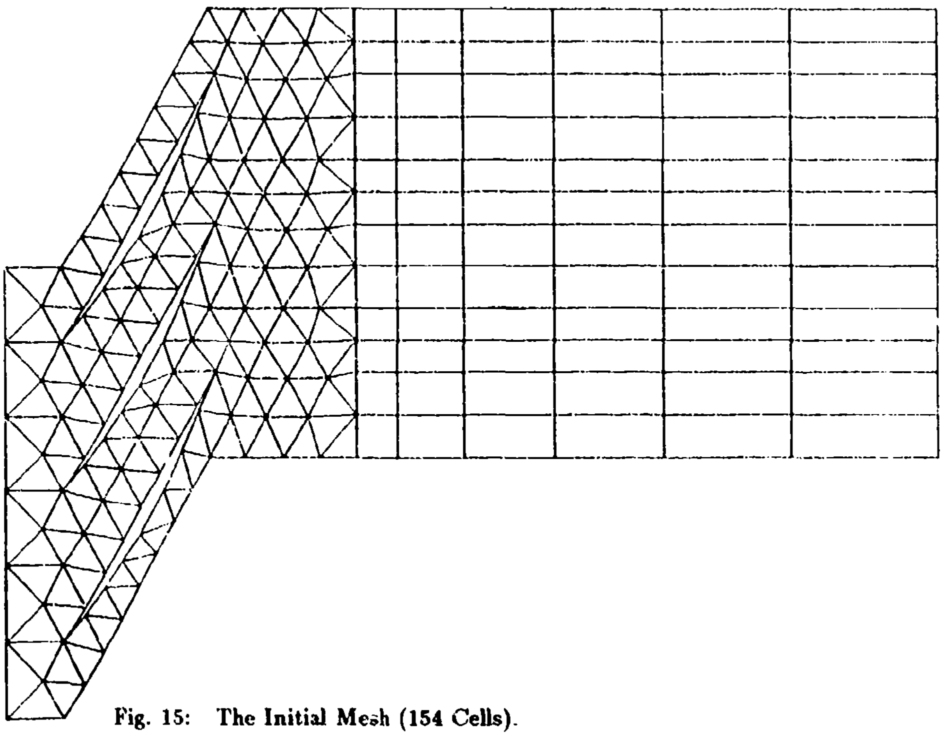


Fig. 15: The Initial Mesh (154 Cells).

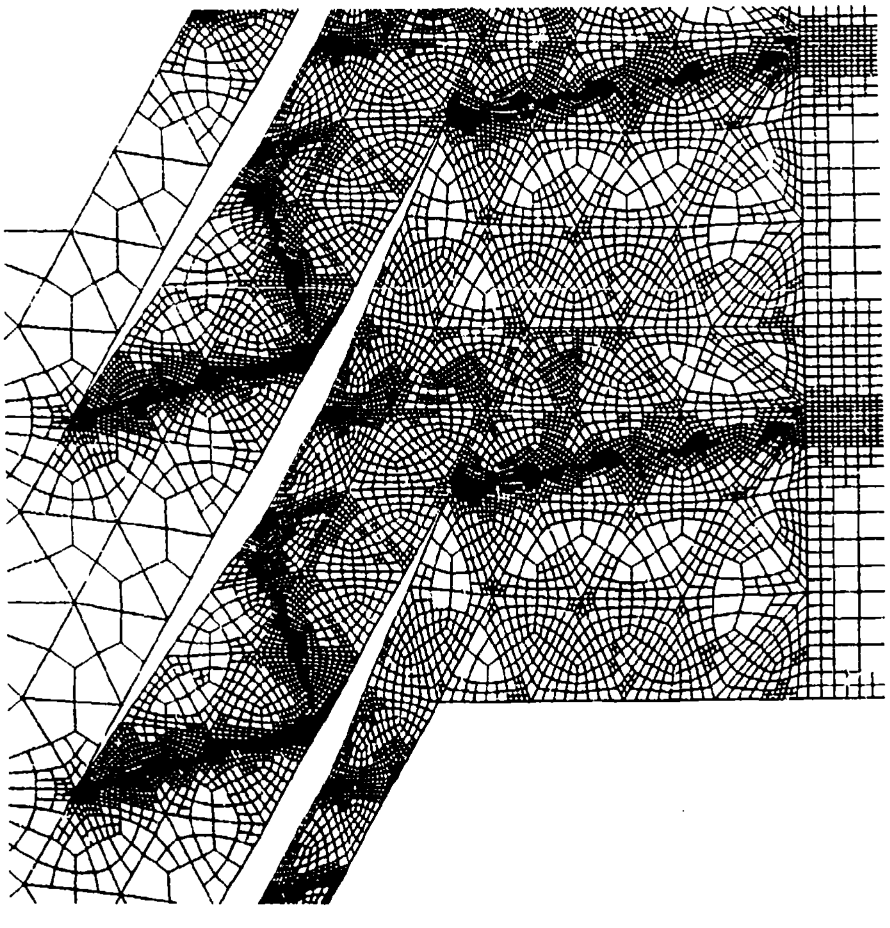


Fig. 16: The Final Mesh (9333 Cells).

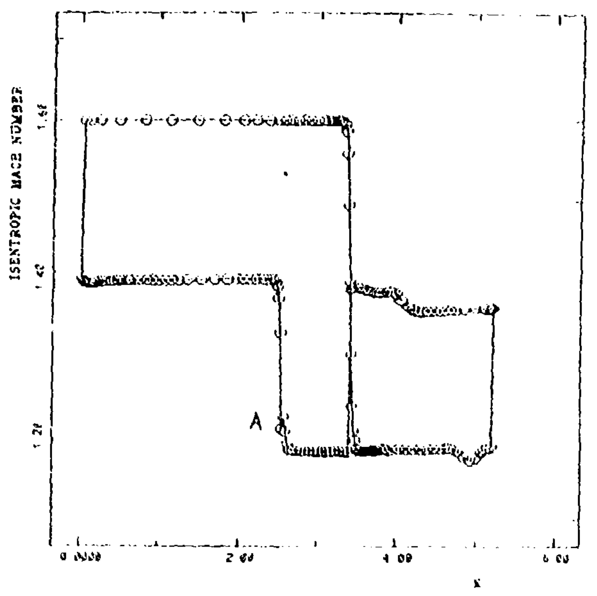


Fig. 17a: Isentropic Mach Number Distribution with Present Method.

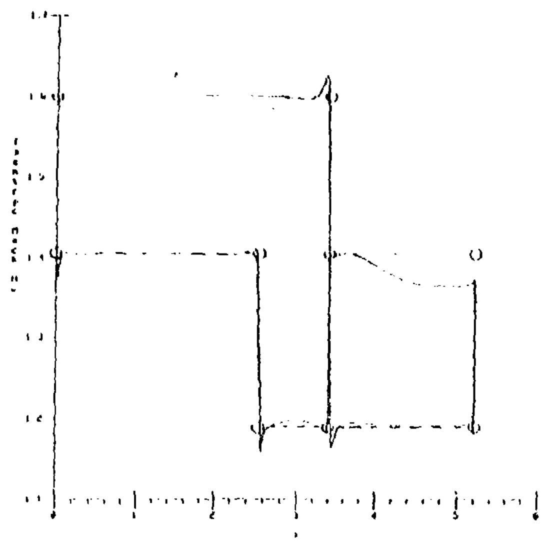


Fig. 17b: Isentropic Mach Number Distribution by Holmes.

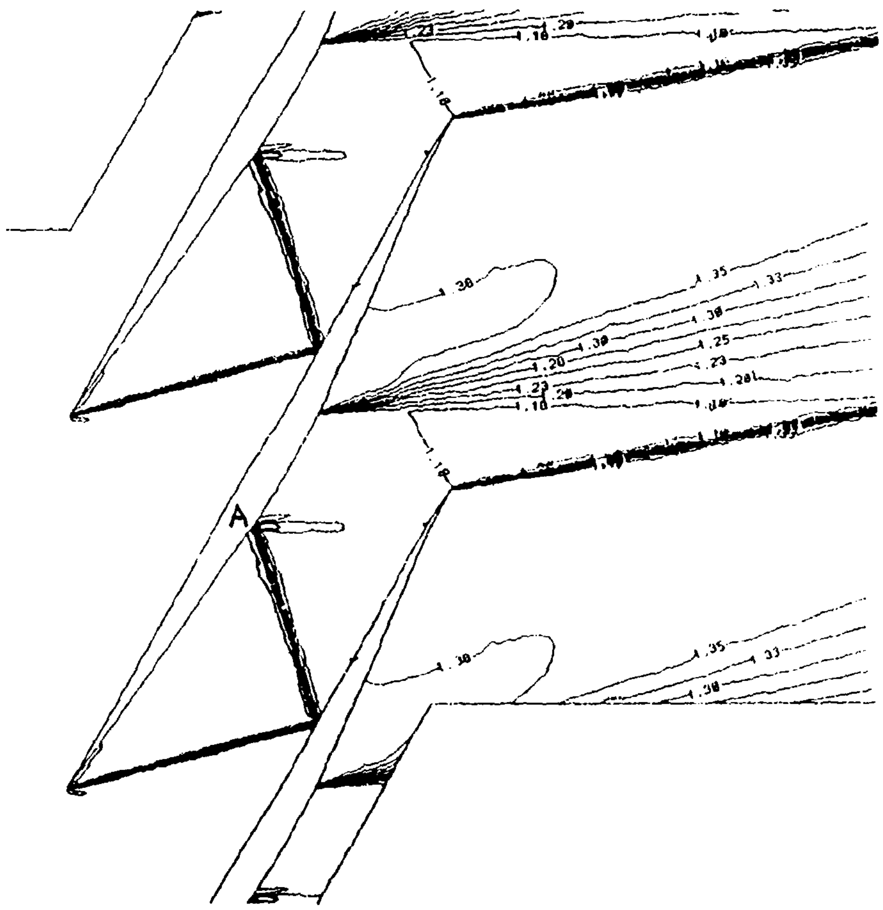


Fig. 18a: Isentropic Mach Number Lines with Present Method.

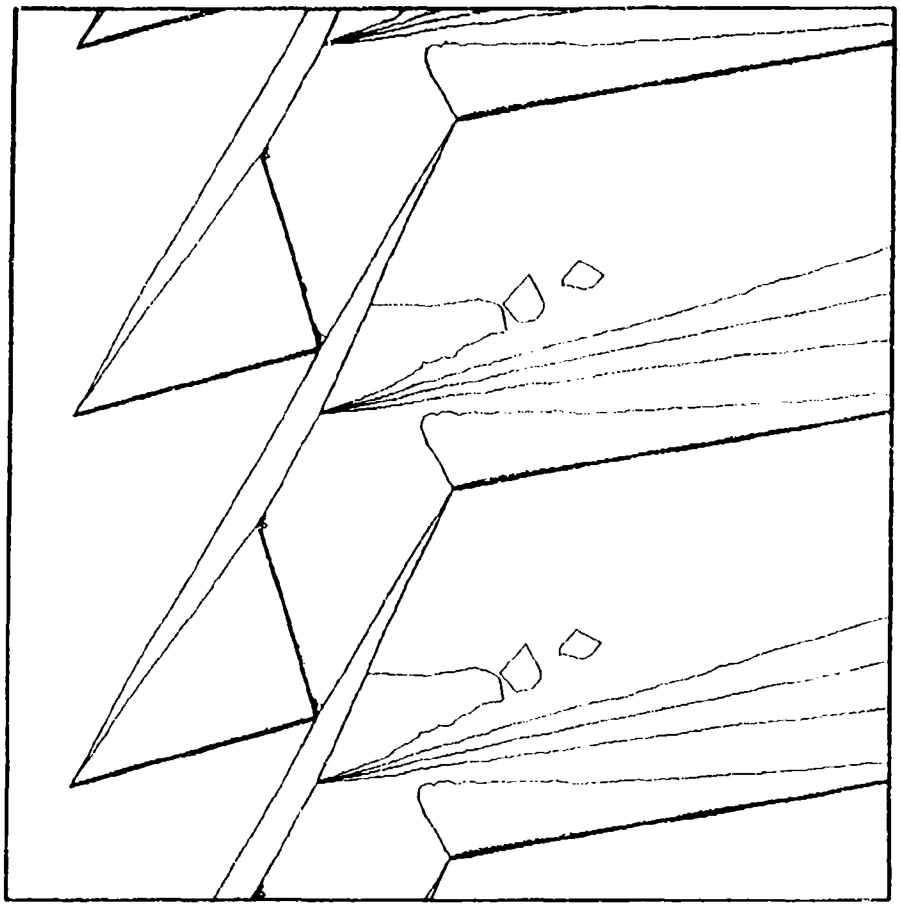


Fig. 18b: Isentropic Mach Number Lines (Holmes).

4 RECONSTRUCTION ALGORITHMS

4.1 Introduction

The estimate of the solution variables at the cell interfaces based on linear extrapolation given by eq. (10) becomes inaccurate for non-linear functions and the idea proposed by Barth [7, 8] is to perform a higher order extrapolation which is still exact for polynomial functions $q(x, y)$ of higher degree over the domain Ω. In turn, this higher order extrapolation will allow for more accurate flux evaluations at the cell interface. A variant of Barth's reconstruction algorithm based on zero-mean basis polynomials (Zero-Mean Higher Order Reconstruction or briefly: ZM-HOR) is presented in this section together with a new reconstruction algorithm using basis polynomials with a zero-value at some reference point of the cell where the solution is to be reconstructed. In general, this reference point will be the gravity center of the cell (GC-HOR). The second algorithm is believed to be computationally less involving since less inertial moments are to be computed.

Note that solution reconstruction is just a way of interpolating a discrete set of given point values or mean values. Other interpolation techniques can be devised such as in Harten [16] whereby the derivatives of a truncated Taylor series expansion are determined by stating that the mean of this series in each support cell must equal the respective given discrete mean value. Harten also developed the GC counterpart of this algorithm.

4.2 ZM-HOR-Algorithm

The aim is to represent the discrete solution Q in each cell P by a polynomial function $Q_P^{(k)}(x, y)$ of degree k where $k \geq 1$. The discrete solution value $\overline{Q}_P$ assigned to cell P is not related to any reference point of the cell, it is supposed to be the mean of the solution distribution over the cell whereas the notation Q_P denotes the discrete solution value at some reference point $\vec{r}_P$ of the cell P. This reference point is in general cell P's gravity center $\vec{r}_{G_P}$. Furthermore, the function $Q_P^{(k)}$ has to satisfy the following requirements, fig. 19:

- it must conserve the mean in the cell P. In other words, the mean value of $Q_P^{(k)}$ over the cell area Ω_P must be equal to the discretized average value $\overline{Q}_P$, assigned to cell P:

$$\overline{Q}_P = \frac{1}{\Omega_P} \cdot \iint_{\Omega_P} Q_P^{(k)} . d\Omega = < Q_P^{(k)}(\vec{r}) >^P \quad (29)$$

- it must represent polynomial functions of degree $r \leq k$ exactly over the whole cell area. In this case, the reconstruction $Q^{(k)}$ over the complete domain will be continuous across the cell edges. If $r > k$ then discontinuities across the cell edges are allowed.

Note that cell P can be either an inter... cell or a boundary cell.

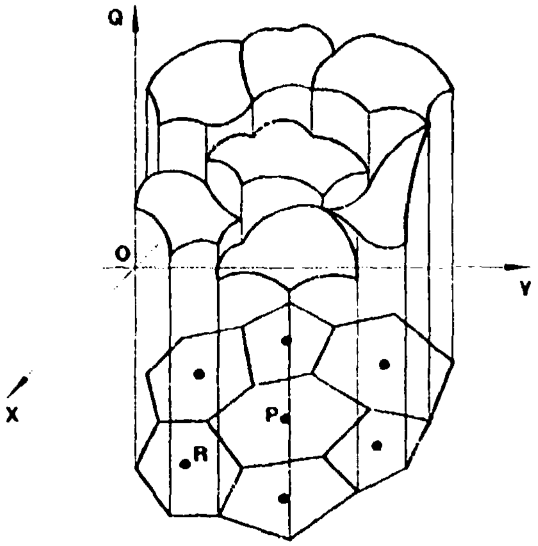


Fig. 19: Representing the solution in cell P with a polynomial function.

In the following, the origin of the coordinate system is always translated towards some reference point $\vec{r}_P$ associated with cell P. The effect of round-off errors in the implementation is attenuated due to this translation of the coordinate system.

Conservation of the mean is met if $Q_P^{(k)}$ belongs to a function space $V_k(\Omega_P)$ with a basis of zero mean polynomials defined by:

$$\mathcal{F} = \{F_{i,j}(\vec{r}) : 0 \leq i + j \leq k \text{ and } \vec{r} \in \Omega_P\}$$

where i and j are always positive and

$$F_{i,j}(\vec{r}) = \begin{cases} 1 & \text{for } i = j = 0 \\ \Delta x_P^i . \Delta y_P^j - I_{i,j}^{P,P} & \text{for } 0 < i + j \leq k \end{cases} \quad (30)$$

with

$$\Delta x_P = x - x_P \quad (31)$$

$$\Delta y_P = y - y_P \quad (32)$$

$$I_{i,j}^{P,P} = \frac{1}{\Omega_P} \cdot \iint_{\Omega_P} \Delta x_P^i . \Delta y_P^j . d\Omega \quad (33)$$

4.2.1 Barth's Version

Barth [7, 8] proposes the following form for the reconstruction polynomial $Q_P^{(k)}(\vec{r})$ in the cell P:

$$\begin{aligned} Q_P^{(k)}(\vec{r}) &= \sum_{i+j=0}^{i+j\leq k} V_{i,j} \, . \, F_{i,j}(\vec{r}) \\ &= \mathbf{V}.\mathbf{F} \end{aligned} \tag{34}$$

with

$$\mathbf{V} = \left[V_{0,0} \cdots V_{i,j} \cdots V_{0,k}\right]^T \tag{35}$$

and

$$\mathbf{F} = \left[F_{0,0}(\vec{r}) \cdots F_{i,j}(\vec{r}) \cdots F_{0,k}(\vec{r})\right]^T \tag{36}$$

The expression $\mathbf{V}.\mathbf{F}$ is the scalar product of the two vectors; i.e. the sum of the products of their corresponding components. Note also that the coordinates of any vector $\vec{r}$ are given in a normalized coordinate system related the cell P.

Barth suggests further the following form for the coefficients in the vector $\mathbf{V}$:

$$\mathbf{V} = \mathbf{W}.\overline{\mathbf{Q}} \tag{37}$$

where

$$\mathbf{W} = \begin{bmatrix} W_{0,0}^{R_1} & W_{0,0}^{R_2} & \cdots & W_{0,0}^{R_n} \\ W_{1,0}^{R_1} & W_{1,0}^{R_2} & \cdots & W_{1,0}^{R_n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{0,k}^{R_1} & W_{0,k}^{R_2} & \cdots & W_{0,k}^{R_n} \end{bmatrix} \tag{38}$$

and

$$\overline{\mathbf{Q}} = \left[\overline{Q}_{R_1} \; \overline{Q}_{R_2} \cdots \overline{Q}_{R_n}\right]^T \tag{39}$$

in which $W_{i,j}^R$ denotes the weight assigned to the support cell R when computing the coefficient $V_{i,j}$ of the reconstruction polynomial in cell P. The subscript n gives the number of cells present in the support for cell P. The support is a set of cells in the neighbourhood of cell P. The array $\overline{\mathbf{Q}}$ contains the discrete solution averages in each support cell.

The weight factors $W_{i,j}^R$ can now be determined by requiring that each of the zero mean basis polynomials $F_{s,t}(\vec{r})$, with $0\leq s+t\leq k$, must be represented exactly by the reconstruction algorithm independently of the local grid shape. In other words, the right hand side of eq. (34) must match exactly with each basis polynomial $F_{s,t}(\vec{r})$ over the cell P if $F_{s,t}(\vec{r})$ is specified as exact cell averages $< F_{s,t}(\vec{r}) >^P$ and $< F_{s,t}(\vec{r}) >^R$ in the cells P and R respectively:

$$\mathbf{F}(\vec{r}) \equiv \mathbf{A} \, . \, \mathbf{F}(\vec{r}) \tag{40}$$

where $\mathbf{A}$ is now a *matrix* whose rows contain the coefficients $A_{i,j}^{s,t}$ ($0\leq i+j\leq k$ and $0\leq s+t\leq k$) to reconstruct the corresponding basis polynomial $F_{s,t}(\vec{r})$ in the left hand side of eq. (40):

$$\mathbf{A} = \begin{bmatrix} A_{0,0}^{0,0} & A_{1,0}^{0,0} & \cdots & A_{0,k}^{0,0} \\ A_{0,0}^{1,0} & A_{1,0}^{1,0} & \cdots & A_{0,k}^{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ A_{0,0}^{0,k} & A_{1,0}^{0,k} & \cdots & A_{0,k}^{0,k} \end{bmatrix} \tag{41}$$

It is easily seen that eq. (40) is satisfied if

$$\mathbf{A} = \mathbf{\Delta} \tag{42}$$

where $\mathbf{\Delta}$ is the $(m \times m)$ unity matrix with m the number of coefficients in a two dimensional polynomial of degree k:

$$m = \frac{(k+1).(k+2)}{2} \tag{43}$$

Knowing that one is reconstructing a known function vector $\mathbf{F}(\vec{r})$ and using eq. (37), this results in the following set of systems of equations:

$$\mathbf{W}.\overline{\mathbf{F}} = \mathbf{\Delta} \tag{44}$$

or

$$\left[\overline{\mathbf{F}}\right]^T . \left[\mathbf{W}\right]^T = \mathbf{\Delta} \tag{45}$$

whereby the matrix $\overline{\mathbf{F}}$ contains the mean $< F_{i,j}(\vec{r}) >^R$ of the known basis polynomials over each support cell R with $0\leq i+j\leq k$:

$$\overline{\mathbf{F}} = \begin{bmatrix} < F_{0,0} >^{R_1} & < F_{1,0} >^{R_1} & \cdots & < F_{0,k} >^{R_1} \\ < F_{0,0} >^{R_2} & < F_{1,0} >^{R_2} & \cdots & < F_{0,k} >^{R_2} \\ \vdots & \vdots & \ddots & \vdots \\ < F_{0,0} >^{R_n} & < F_{1,0} >^{R_n} & \cdots & < F_{0,k} >^{R_n} \end{bmatrix} \tag{46}$$

By construction the elements of $\overline{\mathbf{F}}$ only depend on geometrical characteristics of the support used for the reconstruction in cell P. Therefore, also the weights obtained as a solution of eq. (44) are only geometry dependent. Eq. (45) represents a set of m systems of m equations with n unknowns where m is given by eq. (43) and n is the number of cells R taken into the support. The first system of eq. (45) reads while taking into account that $< F_{0,0} >^R \equiv 1, \; \forall R$:

$$\sum_R W_{0,0}^R = 1 \tag{47}$$

and

$$\sum_R W_{i,j}^R = 0 \;\; , \; 0 < i+j \leq k \tag{48}$$

This reduces the system (44) to:

$$\mathbf{W}.\mathbf{I} = \mathbf{\Delta} \tag{49}$$

in which the matrix $\mathbf{I}$ is given by

$$\mathbf{I} = \begin{bmatrix} I_{0,0}^{R_1,P} & I_{1,0}^{R_1,P} & \cdots & I_{0,k}^{R_1,P} \\ I_{0,0}^{R_2,P} & I_{1,0}^{R_2,P} & \cdots & I_{0,k}^{R_2,P} \\ \vdots & \vdots & \ddots & \vdots \\ I_{0,0}^{R_n,P} & I_{1,0}^{R_n,P} & \cdots & I_{0,k}^{R_n,P} \end{bmatrix} \tag{50}$$

where $I_{i,j}^{R,P}$ is the inertial moment of order (i,j) of the support cells R with respect to the reference point $\vec{r}_P$ of cell P:

$$I_{i,j}^{R,P} = \frac{1}{\Omega_R} . \iint_{\Omega_R} \Delta x_P^i . \Delta y_P^j . d\Omega \tag{51}$$

Remark that $I_{0,0}^{R,P} = 1, \; \forall R$. To be able to solve each system of equations exactly in (49), the number of support cells n should be taken at least equal to m. The selection of the cells R needed for the algorithm will be discussed in a later section.

As n is at least equal to m, each system of eq. (49) is an underdetermined linear system of equations. They can be solved using a modified Gram-Schmidt algorithm with column pivoting, see ref. [27]. The arbitrary constants in the general solution of eq. (49) can be determined by imposing additional constraints on the weights $W_{i,j}^R$. As we want now the reconstruction to be as insensitive as possible to noise in the given data, the norm of the weights must be minimal:

$$\sum_R \left(W_{i,j}^R\right)^2 \text{ must be minimal} \qquad (52)$$

for $0 \le i + j \le k$. In ref. [27], it is shown that the least squares problem (52) is equivalent to solving an overdetermined system of n equations with $n - m$ unknowns (the arbitrary constants in the general solution). A least squares solution of the system can be obtained using again the modified Gram-Schmidt algorithm mentioned above and is then inserted in the general solution of eq. (64).

4.2.2 Present Version

The version of the ZM-HOR-Algorithm presented here differs from Barth's version in two manners. First, no local coordinate transformation is performed towards a normalized system. Secondly, in this approach, the reconstruction is seen as separated into a constant part equal to the given cell average and a part representing the higher order terms:

$$Q_P^{(k)}(\vec{r}) = \overline{Q}_P + [\mathbf{V}.\mathbf{F} - V_{0,0}.F_{0,0}(\vec{r})] \qquad (53)$$

where $\mathbf{V}$ is again the *vector* as in eq. (34) and (35). In other words, the term $V_{0,0}.F_{0,0}(\vec{r})$ in eq. (34) is replaced by the given cell average. Equation (53) can therefore be rewritten as:

$$Q_P^{(k)}(\vec{r}) = \overline{Q}_P + \tilde{\mathbf{V}}.\tilde{\mathbf{F}} \qquad (54)$$

with

$$\tilde{\mathbf{V}} = \left[V_{1,0} \cdots V_{i,j} \cdots V_{0,k}\right]^T \qquad (55)$$

and

$$\tilde{\mathbf{F}} = \left[F_{1,0}(\vec{r}) \cdots F_{i,j}(\vec{r}) \cdots F_{0,k}(\vec{r})\right]^T \qquad (56)$$

Note that the elements with subscript $(0,0)$ are no longer present in eq. (55) and (56).

As in Barth's version, the coefficients in $\tilde{\mathbf{V}}$ in eq. (54) are viewed as a weighted sum of the given discrete averages in the support cells:

$$\tilde{\mathbf{V}} = \mathbf{W}.\overline{\mathbf{Q}} \qquad (57)$$

where now

$$\mathbf{W} = \begin{bmatrix} W_{1,0}^{R_1} & W_{1,0}^{R_2} & \cdots & W_{1,0}^{R_n} \\ W_{2,0}^{R_1} & W_{2,0}^{R_2} & \cdots & W_{2,0}^{R_n} \\ \vdots & \vdots & \ddots & \vdots \\ W_{0,k}^{R_1} & W_{0,k}^{R_2} & \cdots & W_{0,k}^{R_n} \end{bmatrix} \qquad (58)$$

Again, the weight factors $W_{i,j}^R$ with $0 < i + j \le k$ can now be determined by requiring that each of the zero mean basis polynomials $F_{s,t}(\vec{r})$, with $0 \le s + t \le k$, must be represented exactly by the reconstruction algorithm independently of the local grid shape.

$$\mathbf{F}(\vec{r}) \equiv \langle \mathbf{F}(\vec{r}) \rangle^P + \tilde{\mathbf{A}}.\tilde{\mathbf{F}}(\vec{r}) \qquad (59)$$

where, similar to Barth's version, $\tilde{\mathbf{A}}$ is now a $[m \times (m-1)]$ matrix defined by:

$$\tilde{\mathbf{A}} = \begin{bmatrix} A_{1,0}^{0,0} & A_{2,0}^{0,0} & \cdots & A_{0,k}^{0,0} \\ A_{1,0}^{1,0} & A_{2,0}^{1,0} & \cdots & A_{0,k}^{1,0} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1,0}^{0,k} & A_{2,0}^{0,k} & \cdots & A_{0,k}^{0,k} \end{bmatrix} \qquad (60)$$

Eq. (59) is satisfied over the whole cell P if

$$\tilde{\mathbf{A}} = \left[\begin{array}{c|c} 0 & \\ \vdots & \Delta \\ 0 & \end{array}\right] \equiv \mathbf{B} \qquad (61)$$

where Δ is now the $[(m-1) \times (m-1)]$ unity matrix. Knowing that one is reconstructing a known function vector $\mathbf{F}(\vec{r})$ and using eq. (57), this results in the following set of systems of equations:

$$\mathbf{W}.\overline{\mathbf{F}} = \mathbf{B} \qquad (62)$$

whereby the matrix $\overline{\mathbf{F}}$ remains the same as in eq. (46). Note that the components of $\vec{r}$ are still related to some reference point $\vec{r}_P$ associated with the cell P.

Since $< F_{0,0}(\vec{r}) >^R \equiv 1$, the first system of eq. (62) leads now to:

$$\sum_R W_{i,j}^R = 0 \quad , 0 < i + j \le k \qquad (63)$$

This reduces the system (62) to:

$$\mathbf{W}.\mathbf{I} = \mathbf{B} \qquad (64)$$

or

$$\mathbf{I}^T.\mathbf{W}^T = \mathbf{B}^T \qquad (65)$$

where the matrix I is defined in eq. (50). Equation (65) represents a set of $m-1$ systems of m equations with n unknowns where n is the number of support cells and m is given by eq. (43). Solving this set of systems is done in exactly the same way as in Barth's version except that the number of systems to solve is reduced by one.

4.2.3 Conservation of the Mean

We will now investigate to what extent both versions (34) and (54) of the ZM-HOR-Algorithm conserve the mean of a solution function $Q(\vec{r})$ specified as a discrete set of cell averages $\overline{Q}_P$ whereby:

$$\overline{Q}_P \equiv < Q >^P \qquad (66)$$

As zero mean polynomials are used, it follows immediately that the mean of eq. (54) is equal to the given cell average. This means that the present version always conserves the mean even if the degree of the function to reconstruct is beyond k.

Taking the mean of Barth's formulation of the reconstruction polynomial in eq. (34), one finds:

$$< Q_P^{(k)}(\vec{r}) >^P = V_{0,0} = \sum_R W_{0,0}^R \overline{Q}_R \qquad (67)$$

Let us now expand Q in a Taylor series and average it over each support cell R. Therefore, two vectors are now introduced:

$$\mathbf{I}_l^{R,P} = \left[I_{l,0}^{R,P} \cdots I_{l-m,m}^{R,P} \cdots I_{0,l}^{R,P}\right] \qquad (68)$$

containing the inertial moments of cell R of exactly order l with respect to the reference point $\vec{r}_P$ of cell P and

$$\partial \mathbf{Q}_l^P = \begin{bmatrix} \binom{l}{0} .\partial Q_{l,0}^P \\ \vdots \\ \binom{l}{m} .\partial Q_{l-m,m}^P \\ \vdots \\ \binom{l}{l} .\partial Q_{0,l}^P \end{bmatrix} \tag{69}$$

with

$$\partial Q_{l-m,m}^P = \left. \frac{\partial^l Q}{\partial x^{l-m} \partial y^m} \right|_P \quad , \quad 0 \leq m \leq l \tag{70}$$

the derivatives of order l of the function Q at the reference point $\vec{r}_P$ of cell P. One can then average the Taylor series expansion of $\overline{Q}_P$ and $\overline{Q}_R$ around the reference point $\vec{r}_P$ giving (note that the operators $< \cdot >^P$ and $< \cdot >^R$ are linear):

$$\begin{aligned} \overline{Q}_P &\equiv < Q >^P \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} < \left\{ \left[(\vec{r} - \vec{r}_P).\vec{\nabla} \right]^l Q \right\}_P >^P \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} \sum_{m=0}^{l} I_{l-m,m}^{P,P} .\partial Q_{l-m,m}^P \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} \mathbf{I}_l^{P,P} .\partial \mathbf{Q}_l^P \end{aligned} \tag{71}$$

and

$$\begin{aligned} \overline{Q}_R &\equiv < Q >^R \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} < \left\{ \left[(\vec{r} - \vec{r}_P).\vec{\nabla} \right]^l Q \right\}_P >^R \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} \sum_{m=0}^{l} I_{l-m,m}^{R,P} .\partial Q_{l-m,m}^P \\ &= Q_P + \sum_{l=1}^{\infty} \frac{1}{l!} \mathbf{I}_l^{R,P} .\partial \mathbf{Q}_l^P \end{aligned} \tag{72}$$

where $Q_P = Q(\vec{r}_P)$ and $Q_R = Q(\vec{r}_R)$. Inserting the above equation in eq. (67) while using also eq. (48) leads to:

$$\begin{aligned} < Q_P^{(k)}(\vec{r}) >^P &= Q_P + \sum_{l=1}^{k} \frac{1}{l!} \sum_R W_{0,0}^R . \left(\mathbf{I}_l^{R,P} .\partial \mathbf{Q}_l^P \right) \\ &+ \mathcal{O}(h^{k+1}) \mathcal{D}_Q^{k+1} \\ &= Q_P + \sum_{l=1}^{k} \frac{1}{l!} \left(\sum_R W_{0,0}^R . \mathbf{I}_l^{R,P} \right) .\partial \mathbf{Q}_l^P \\ &+ \mathcal{O}(h^{k+1}) \mathcal{D}_Q^{k+1} \end{aligned} \tag{73}$$

The expression $\mathcal{D}_Q^{k+1}$ stands for space derivatives of order $k+1$ of Q and h is a typical length associated with cell P; e.g. $h = \sqrt{\Omega_P}$. Knowing that l is never zero in the summation of eq. (73) and applying eqs. (49), one finds that the summation over l is equal to zero and one gets using eq. (71):

$$\begin{aligned} < Q_P^{(k)}(\vec{r}) >^P &= Q_P + \mathcal{O}(h^{k+1}) \mathcal{D}_Q^{k+1} \\ &= \overline{Q}_P + \mathcal{O}(h) \mathcal{D}_Q^1 \end{aligned}$$

Hence, Barth's version of the ZM-HOR-Algorithm only conserves the mean to order h, even if the given function Q were a polynomial of degree $r \leq k$. In that case, the mean of the construction would exactly equal the value Q_P of the function Q in the reference point of cell P. As the reference point is in most cases equal to the centroid of the cell and as Q is not always linear, Q_P is in general not equal to $\overline{Q}_P$. However, conservation of the mean will be established in the limit of $h \to 0$.

4.3 The GC-HOR-Algorithm

The discrete solution Q_P of cell P is now supposed to be given at some reference point $\vec{r}_P$ associated with the cell P:

$$Q_P = Q(\vec{r}_P) \tag{74}$$

In practice, the reference point will coincide with the gravity center of the cell P.
The concept of a generic reference point for the reconstruction algorithm is also very useful for studying cell-vertex schemes for which the discretized solution is in general not stored at the gravity center of the associated control volume. However, in the frame of this contribution, the ideas are only fixed on cell-centered storage.

The aim is again to represent the solution in each cell P by a polynomial function $Q_P^{(k)}(\vec{r})$ of degree k where $k \geq 1$. The cell P can either be an interior cell or a boundary cell. The function $Q_P^{(k)}$ has to satisfy the following requirements, fig. 19:

- its value at the reference point $\vec{r}_P(x_P, y_P)$ of cell P must be equal to the discrete solution value assigned to the cell P:
$$Q_P^{(k)}(\vec{r}_P) = Q_P \tag{75}$$
- it must represent polynomial functions of degree $r \leq k$ exactly over the whole cell area. In this case, the reconstruction $Q^{(k)}$ over the complete domain will be continuous across the cell edges. If $r > k$ then discontinuities across the cell edges are allowed.

Again, the origin of the coordinate system is always translated towards the reference point $\vec{r}_P$ of the cell P in order to reduce the influence of round-off error in the algorithm. Requirement (75) is met if $Q_P^{(k)}$ belongs to a function space $V_k(\Omega_P)$ where its basis is given by

$$\mathcal{G} = \{ G_{i,j}(\vec{r}) : 0 \leq i + j \leq k \text{ and } \vec{r} \in \Omega_P \}.$$

where now the basis functions $G_{i,j}(\vec{r})$ are given by

$$G_{i,j} = \Delta x_P^i . \Delta y_P^j : 0 \leq i + j \leq k \tag{76}$$

As for the ZM-HOR-Algorithm, the reconstruction polynomial is given by a linear combination of the basis polynomials:

$$Q_P^{(k)}(\vec{r}) = \mathbf{V}.\mathbf{G} \tag{77}$$

where the vector $\mathbf{V}$ is given by eq. (55) and the vector $\mathbf{G}$ is similar to the vector $\mathbf{F}$ defined in eq. (56) but uses the non-zero mean basis polynomials $G_{i,j}$:

$$\mathbf{G} = \left[G_{1,0}(\vec{r}) \cdots G_{i,j}(\vec{r}) \cdots G_{0,k}(\vec{r}) \right]^T \tag{78}$$

Also here, the coefficients in the vector $\mathbf{V}$ in eq. (77) are written as a weighted sum of the given discrete values Q_R in a set of cells in the neighbourhood of the cell P:

$$\mathbf{V} = \mathbf{W}.\mathbf{Q} \tag{79}$$

in which

$$\mathbf{Q} = \left[Q_{R_1}\ Q_{R_2}\ \cdots\ Q_{R_n}\right]^T \qquad (80)$$

A similar reasoning as for the ZM-HOR algorithm is now followed leading to the next set of systems of equation for the reconstruction weights:

$$\mathbf{W}.\mathbf{D} = \mathbf{\Delta} \qquad (81)$$

in which the matrix **D** is defined as:

$$\mathbf{D} = \begin{bmatrix} G_{0,0}^{R_1} & G_{1,0}^{R_1} & \cdots & G_{0,k}^{R_1} \\ G_{0,0}^{R_2} & G_{1,0}^{R_2} & \cdots & G_{0,k}^{R_2} \\ \vdots & \vdots & \ddots & \vdots \\ G_{0,0}^{R_n} & G_{1,0}^{R_n} & \cdots & G_{0,k}^{R_n} \end{bmatrix} \qquad (82)$$

with $G_{i,j}^{g,R} = G_{i,j}(\vec{r}_R)$ the point value of a basis polynomials at the reference point of a support cell R. Remark again that eq. (81) gives the following equation for reconstructing constant basis polynomial:

$$\sum_R W_{i,j}^R = 0 \text{ for } 0 < i+j \le k \qquad (83)$$

The set of systems (81) is solved just as it was solved in the case of the zero mean polynomials while adding the constraint of minimal norm.

4.4 Example

Suppose one wants to perform a linear reconstruction using the GC-HOR algorithm in the cell P of a regular orthogonal rectangular mesh with mesh size $\Delta x = \Delta y = 1$, see fig. 20. The reference point of each cell is taken at the gravity center. The support for the reconstruction is chosen to contain the cell P itself and its 4 direct neighbours.

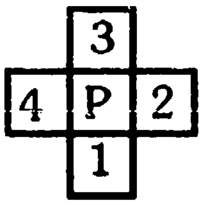


Fig. 20: Support for Linear Reconstruction on a Regular Mesh.

The matrix **D** in eq. (81) becomes then:

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \qquad (84)$$

The first column of **D** corresponds to eq. (83), the second column gives the coefficients of the weights for the term in y and the third column gives the coefficients of the weights for the term in x. The first row is related to cell P while row j is related the neighbour $j-1$ for $j = 2,3,4,5$. The right hand side of the system is given by:

$$\mathbf{RHS} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (85)$$

The solution **W** of the system is easily obtained:

$$\mathbf{W} = \begin{bmatrix} 0 & -1/2 & 0 & 1/2 & 0 \\ 0 & 0 & 1/2 & 0 & -1/2 \end{bmatrix} \qquad (86)$$

in which the first row gives the weights contributing to the term in y and the second row gives the weights contributing to the term in x.
The reconstruction of the solution Q in cell P can then be written as:

$$\begin{aligned} Q_P^{(1)}(x,y) &= Q_P + \frac{Q_2 - Q_4}{2}(x - x_P) \\ &\quad + \frac{Q_3 - Q_1}{2}(y - y_P) \end{aligned} \qquad (87)$$

From which it is found that the gradient of $Q_P^{(1)}$ is given by the classic central approximation formulas.

As the value of a linear function in the gravity center of a cell is also equal to the mean value of the linear function over the cell, it can be shown easily that the ZM-HOR-algorithm gives the same reconstruction weights when performing a first order reconstruction.

4.5 Caveat

If one takes a support as shown in fig. 21 in order to perform a second order reconstruction, then one finds that all coefficients in the matrix A of the weights contributing to the term in xy are zero. Note that always either Δx or Δy is zero for each of the elements of the support depicted in fig. 21.

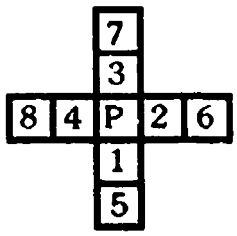


Fig. 21: Erratic Support for Quadratic Reconstruction.

The product of both will therefore always be zero, see also eq. (81) with $s = t = 1$. As in the *RHS* one finds a unit block matrix, the equation with $i = j = 1$ will be contradictory since it states that 0 should equal 1. Therefore, second order polynomial reconstruction on a regular rectangular mesh always requires taking elements along the diagonal of cell P into the support.

Note that this problem also occurs when using the ZM-HOR-algorithm. The mean of $F_{1,1}(x,y)$ will be zero over each of the support cells since $F_{1,1}$ is an uneven function being integrated over a domain with symmetric boundaries.

In general, one can say that a contradictory system is expected if there are not enough cells in all space directions. A contradictory system is avoided if, see also eqs. (49),(64),(81):

$$\text{rank}\, \mathbf{I}^T = m \text{ or } \text{rank}\, \mathbf{D}^T = m \qquad (88)$$

where m is given by eq. (43). The condition can be satisfied by choosing an oversized stencil which obviously increases the computational load. In this work, the stencil is in general oversized by 2. In ref. [16], Harten proposes other techniques for selecting the support in order to obtain a sufficient number of linearly independent equations. However, these techniques are all strongly linked with the chosen reconstruction algorithm.

4.6 Accuracy

Theorem 4.1
If a given solution function Q varies smoothly and if Q is specified as a discrete set of cell averages in each cell, then the ZM-HOR-Algorithm given by eq. (54) approximates a given solution function Q to order h^{k+1} with k the degree of the reconstruction polynomials.

Proof of Theorem 4.1:
As Q varies smoothly, one can then average the Taylor series expansion as given in eq. (72) using the definitions in eq. (68) and (69). Inserting the expression of eq. (72) for $\overline{Q}_R$ in eq. (54) yields:

$$\begin{aligned} Q_P^{(k)}(\vec{r}) &= \overline{Q}_P \\ &+ \sum_{i+j>0}^{i+j\le k} F_{i,j} \sum_{l=1}^{k} \frac{1}{l!}\left(\sum_R W_{i,j}^R . \mathbf{I}_l^{R,P}\right) . \partial\mathbf{Q}_l^P \\ &+ \mathcal{O}(h^{k+2}).\mathcal{D}_Q^{k+1} \end{aligned}$$

Let us now define the vector $\mathbf{F}_l$ as:

$$\mathbf{F}_l = \left[F_{l,0} \ \cdots \ F_{l-m,m} \ \cdots \ F_{0,l}\right] \tag{89}$$

i.e. the vector containing all basis functions of exactly degree l.
Using eq. (64) and expanding also $\overline{Q}_P$ while truncating the series at $l = k$, one gets:

$$\begin{aligned} Q_P^{(k)}(\vec{r}) &= Q_P + \sum_{l=1}^{k} \frac{1}{l!}\mathbf{I}_l^{P,P} . \partial\mathbf{Q}_l^P \\ &+ \sum_{l=1}^{k} \frac{1}{l!}\mathbf{F}_l . \partial\mathbf{Q}_l^P + \mathcal{O}(h^{k+2}).\mathcal{D}_Q^{k+1} \end{aligned}$$

where $\mathcal{D}_Q^{k+1}$ stands for space derivatives of order $k+1$ of Q and h is a typical length associated with cell P; e.g. $h = \sqrt{\Omega_P}$. The m^{th} element of the vector $\mathbf{C}_l^P$ is given by:

$$C_{l-m,m}^P = (x - x_P)^{l-m} . (y - y_P)^m \tag{90}$$

Remark then also that due to eq. (30):

$$\mathbf{F}_l = \mathbf{C}_l - \mathbf{I}_l^{P,P} \tag{91}$$

The reconstruction polynomial reads then:

$$Q_P^{(k)}(\vec{r}) = Q_P + \sum_{l=1}^{k} \frac{1}{l!}\mathbf{C}_l^P . \partial\mathbf{Q}_l^P + \mathcal{O}(h^{k+2})\, \mathcal{D}_Q^{k+1}$$

As the Taylor series expansion of $Q(\vec{r})$ around the reference point $\vec{r}_P$ is given by:

$$\begin{aligned} &Q(\vec{r}) \\ &= Q_P + \sum_{l=1}^{k} \frac{1}{l!}\left\{\left[(\vec{r} - \vec{r}_P).\vec{\nabla}\right]^l Q\right\}_P \\ &= Q_P + \sum_{l=1}^{k} \frac{1}{l!} \sum_{m=0}^{l} C_{l-m,m}^P \partial Q_{l-m,m}^P \\ &\quad + \mathcal{O}(h^{k+1}).\mathcal{D}_Q^{k+1} \\ &= Q_P + \sum_{l=1}^{k} \frac{1}{l!}\mathbf{C}_l^P . \partial\mathbf{Q}_l^P + \mathcal{O}(h^{k+1}).\mathcal{D}_Q^{k+1} \end{aligned}$$

one finds that:

$$Q_P^{(k)}(\vec{r}) = Q(\vec{r}) + \mathcal{O}(h^{k+1}).\mathcal{D}_Q^{k+1} \tag{92}$$

□

A similar reasoning for Barth's version of the ZM-HOR-Algorithm leads to the next theorem:

Theorem 4.2
If a given solution function Q varies smoothly and if Q is specified as a discrete set of cell averages in each cell, then the ZM-HOR-Algorithm given by eq. (34) approximates a given solution function Q to order h^{k+1} with k the degree of the reconstruction polynomials.

We will now proof the following theorem concerning the accuracy of the GC-HOR-Algorithm:

Theorem 4.3
If a given solution function Q varies smoothly and if Q is specified as a discrete set of cell averages in each cell, then the GC-HOR-Algorithm given by eq. (77) approximates a given solution function Q to order h^{k+1} with k the degree of the reconstruction polynomials.

Proof of Theorem 4.3:
In a similar way as for the ZM-HOR-Algorithm, one performs a Taylor series expansion around the reference $\vec{r}_P$ of Q_R in the expression for $V_{i,j}$ in eq. (77). Using the properties of the weights $W_{i,j}^R$ expressed by eq. (81) and (83), one finds again:

$$Q_P^{(k)}(\vec{r}) = Q(\vec{r}) + \mathcal{O}(h^{k+1}).\mathcal{D}_Q^{k+1} \tag{93}$$

□

4.7 Results

In the following, some reconstructions of known polynomials using the GC-HOR-algorithm are shown. Similar results were obtained with the ZM-HOR-algorithm. A numerical error study on the higher order reconstruction algorithm was carried out whereby the error between the reconstruction and the exactly known polynomial is examined. The next polynomials were rather arbitrarily selected to perform the numerical error study:

$$\begin{aligned} P_1(x,y) &= 2x + 3y + 2 \\ P_2(x,y) &= \tfrac{1}{4}x^2 + xy \\ &\quad - x - y + \tfrac{5}{4} \\ P_3(x,y) &= \tfrac{1}{16}x^3 + 11y^3 + \tfrac{1}{16}x^2 \\ &\quad - \tfrac{19}{2}y^2 - \tfrac{1}{4}xy + 10 \\ P_4(x,y) &= x^4 + 16y^4 + 8xy^3 \\ &\quad - 34x + 10 \end{aligned} \tag{94}$$

Figures 22, 23, 24 and 25 show an isovalue plot of the four polynomials on a given rectangular domain.
Three different meshes were used on the domain to reconstruct the previous polynomials using the Non-Zero Mean HOR-algorithm while taking the gravity center as a reference point: a) a regular mesh (fig. 26a), b) as a) but twice as fine (fig. 26b) and c) an irregular mesh with

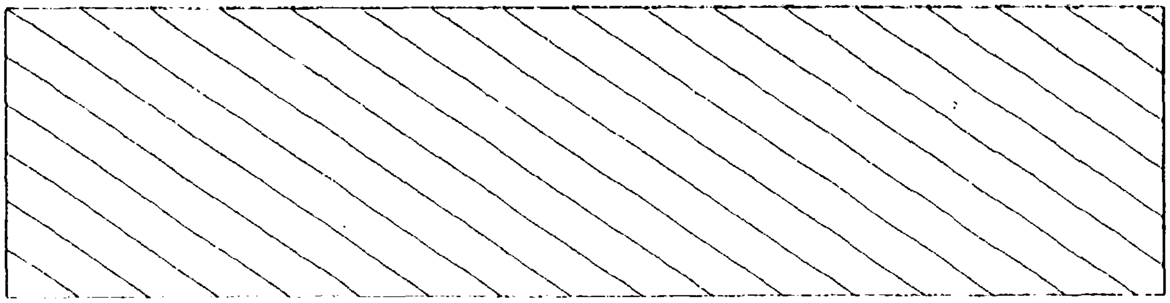


Fig. 22: Isolines of Test Polynomial $P1$ for the GC-HOR-Algorithm.

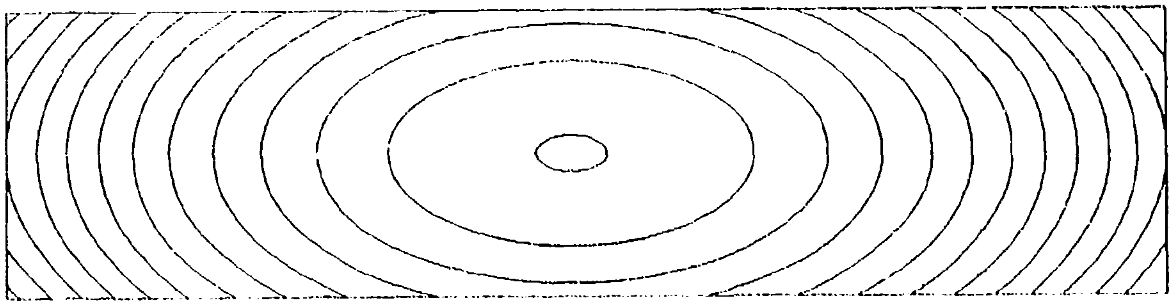


Fig. 23: Isolines of Test Polynomial $P2$ for the GC-HOR-Algorithm.

the same number of points as in a) but they are moved randomly (fig. 26c).

The values of the reconstructed polynomials were then compared with the exact values of the polynomials. The graphs 27a and 27b show the L_2-norm of the error made all over the domain as a function of the order of the reconstruction and this for each of the four given polynomials. The L_2-norm is given by:

$$L_{2,err} = \sqrt{\iint_{\Omega} (Q_P^{(k)} - Q)^2 . d\Omega} \qquad (95)$$

in which $Q_P^{(k)}$ is the reconstructed function, Q the given polynomial function. Eq. (95) is discretized by covering the domain with a fine point cloud. This cloud is triangulated and the error is assumed to vary linearly in each triangle. The integral over the whole domain is then the sum of the integrals over each triangle. As about 13 points were taken in each grid cell, the discretization error will be relatively small (typically 0,1 %).

Comparing the L_2-norms in fig. 27a for the coarse and the fine mesh, one can conclude that the reconstruction error is, as expected, roughly of order $O(h^{k+1})$, if $r > k$. This dependency on the grid spacing is valid for all polynomial degrees r larger than k. Note also that the error is hardly affected by the grid irregularity, as can be deduced form figs. 27a and 27b. From these two figures, it also obvious that the reconstruction error is indeed increasing for a given k when r is increasing. .

In order to see how the errors are distributed over the domain, the relative L_2-norm of the difference between the two functions over each individual cell is computed as follows:

$$\frac{L_{2,err}}{L_{2,Q}} = \frac{\sqrt{\iint_{\Omega} (Q_P^{(k)} - Q)^2 . d\Omega}}{\sqrt{\iint_{\Omega} Q^2 . d\Omega}} \qquad (96)$$

This equation is discretized in exactly the same as eq. (95) but now only over each cell individually.

Making a plot of the relative L_2-norm of the error would only produce random noise if $r \leq k$. Therefore, it makes only sense to examine the error distribution over the domain when $r > k$. Figures 28a, 28b and 28c show the relative error distributions for the polynomials $P2$, $P3$ and $P4$ when using a first order reconstruction on the irregular mesh. It is observed that the largest errors occur where the isolines of the polynomial exhibit a high curvature or where the distance between the isolines is varying strongly. The effect of the terms of higher degree in the polynomials P_2, P_3 and P_4 is stronger in these locations; therefore, the linear reconstruction cannot match anymore with the imposed polynomial and the error increases. The same behaviour is seen on the two other meshes and for the higher order reconstructions as long as $r > k$.

Finally, fig. 29a, 29b and 29c present the linear, quadratic and respectively the cubic reconstruction of the function P_3 on an irregular mesh. These figures demonstrate the increasing quality of the reconstruction with increasing reconstruction order. Again, an exact match is found when the order of the reconstruction equals the degree of the polynomial.

4.8 Remark

Solution reconstruction is just a way of interpolating a given set of discrete values. Hence, it is clear that, besides polynomial basis functions, other basis functions can be

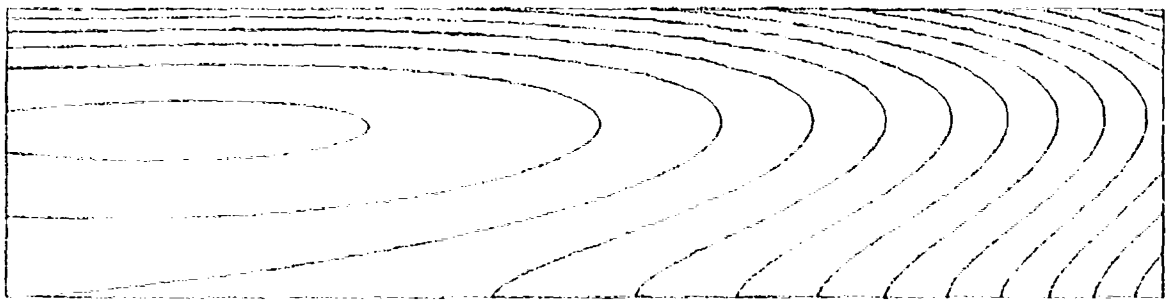


Fig. 24: Isolines of Test Polynomial $P3$ for the GC-HOR-Algorithm.

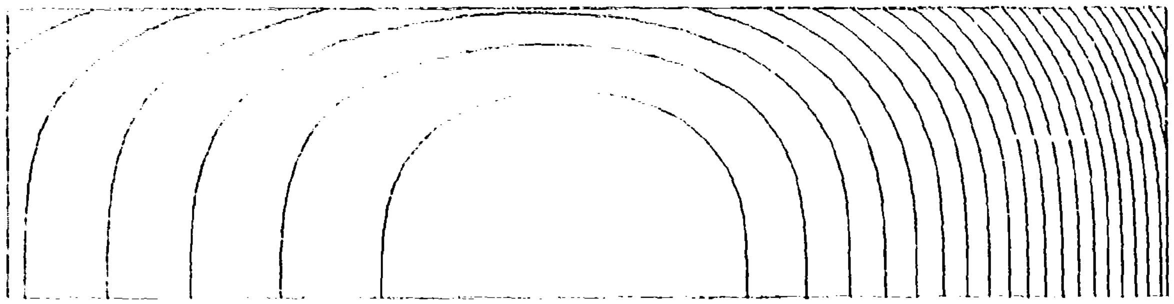


Fig 25: Isolines of Test Polynomial $P4$ for the GC-HOR-Algorithm.

used for solution reconstruction such as trigonometric, hyperbolic or Bessel functions. Unlike in spectral methods, the interpolants need only to be cellwise continuous; i.e. discontinuities are allowed across cell interfaces.
Another interesting interpolation technique was proposed by De Meyer et al. [28, 29], whereby a mixture of trigonometric and polynomial functions is used. Research is still to be done to generalize this technique to finite volume schemes in multiple dimensions.

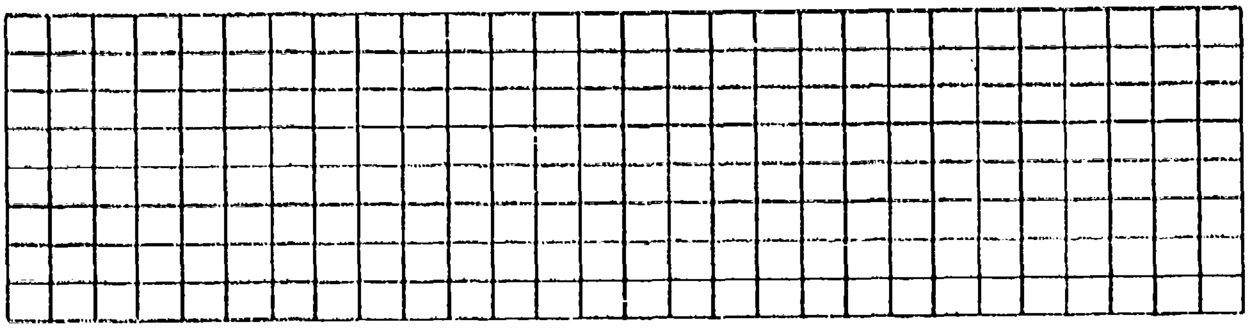


Fig. 26a: Regular Mesh for Testing the GC-HOR-Algorithm.

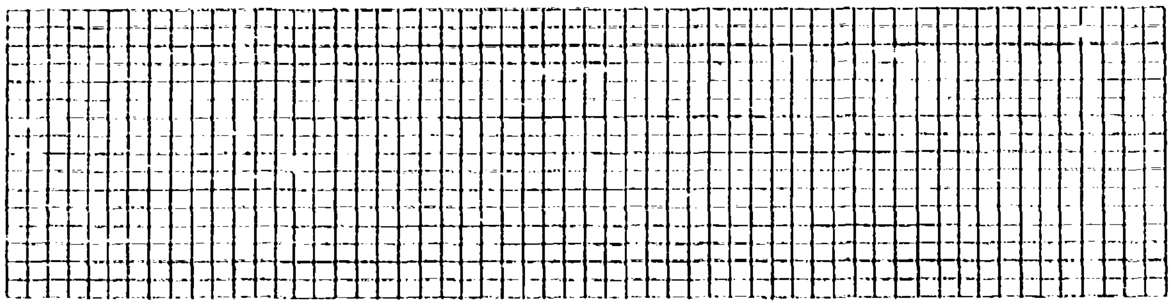


Fig. 26b: Refined Regular Mesh for Testing the GC-HOR-Algorithm.

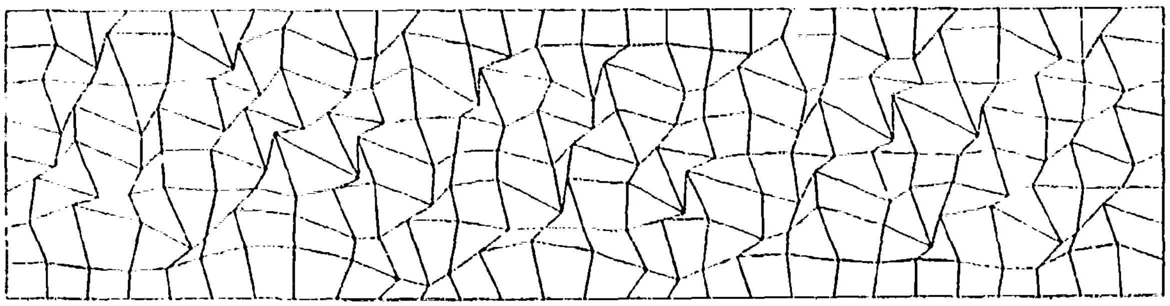


Fig. 26c: Irregular Mesh for Testing the GC-HOR-Algorithm.

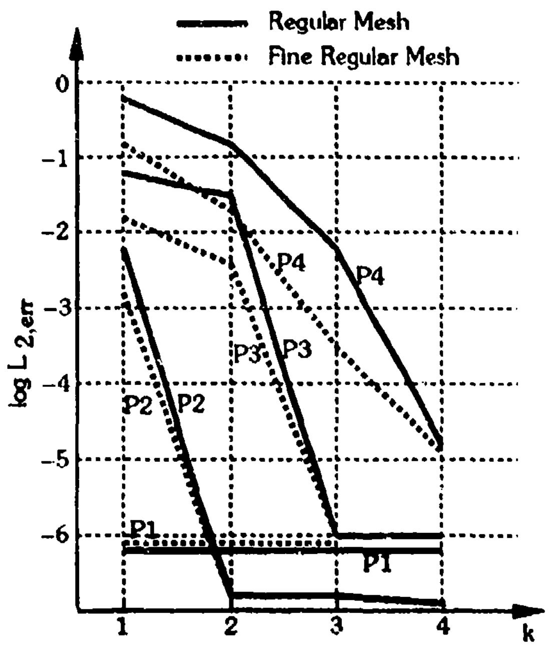


Fig. 27a: Graph of L_2-norm of the Reconstruction Error, Regular and Refined Mesh (GC-HOR).

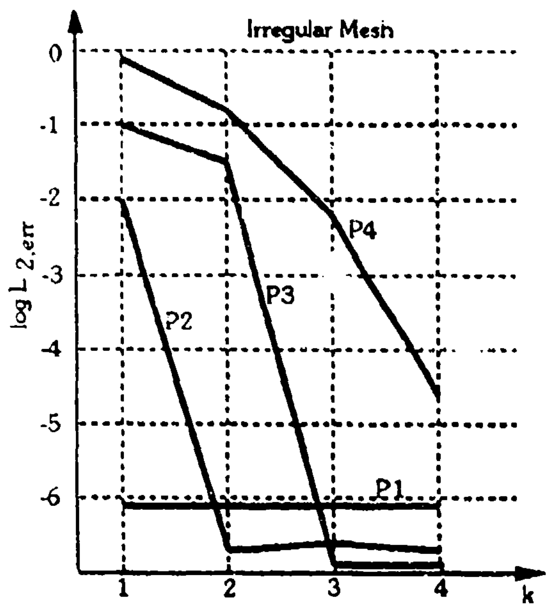


Fig. 27b: Graph of L_2-norm of the Reconstruction Error, Irregular Mesh (GC-HOR).

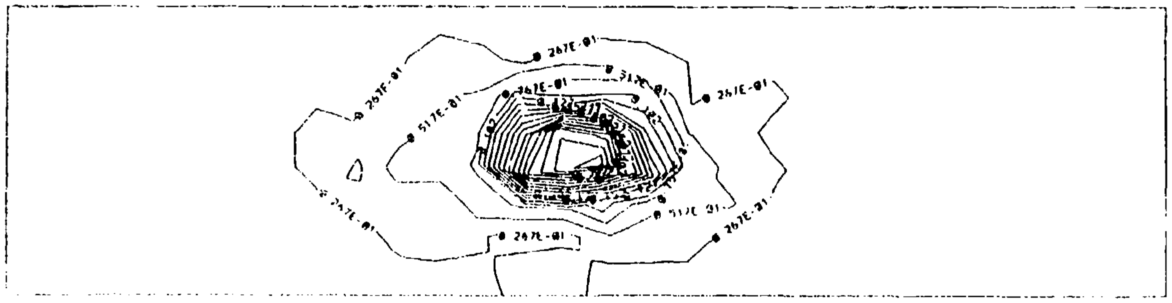


Fig. 28a: Error Distribution for 1st Order Reconstruction of P_2 (GC-HOR), cellwise L_2-norms.

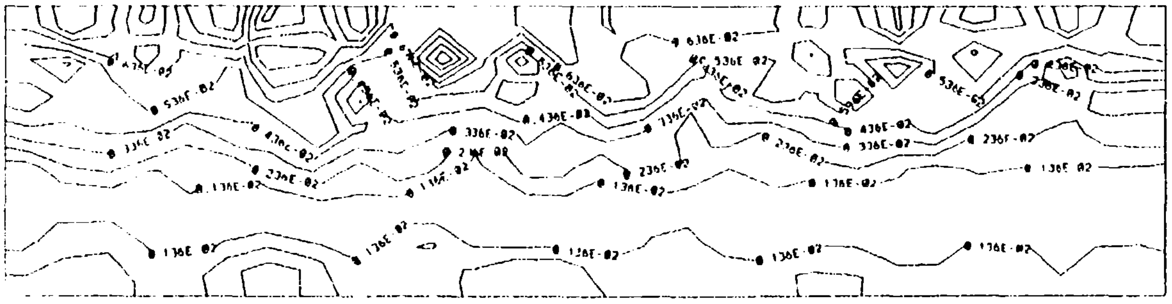


Fig. 28b: Error Distribution for 1st Order Reconstruction of P_3 (GC-HOR), cellwise L_2-norms.

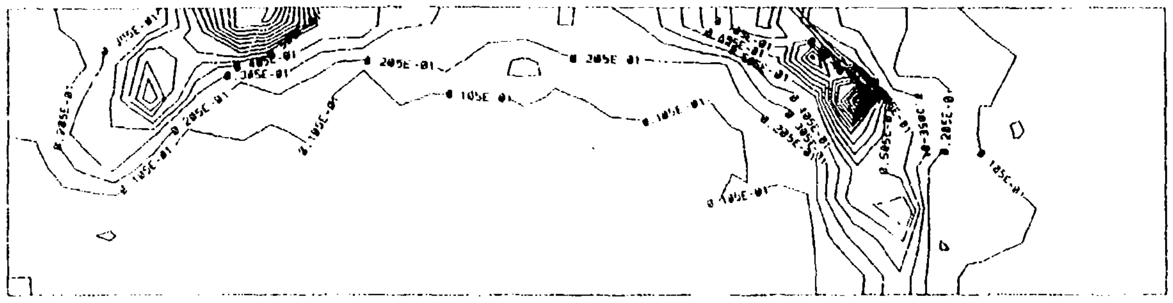


Fig. 28c: Error Distribution for 1st Order Reconstruction of P_4 (GC-HOR), cellwise L_2-norms.

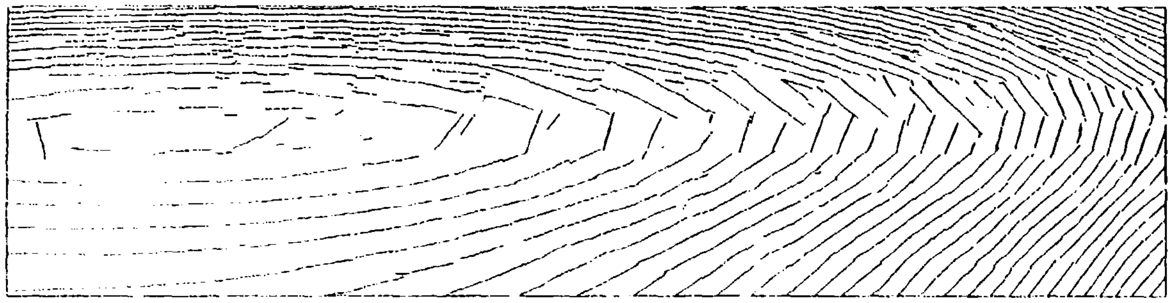


Fig. 29a: 1st Order Reconstruction of P_3, Irregular Mesh (GC-HOR).

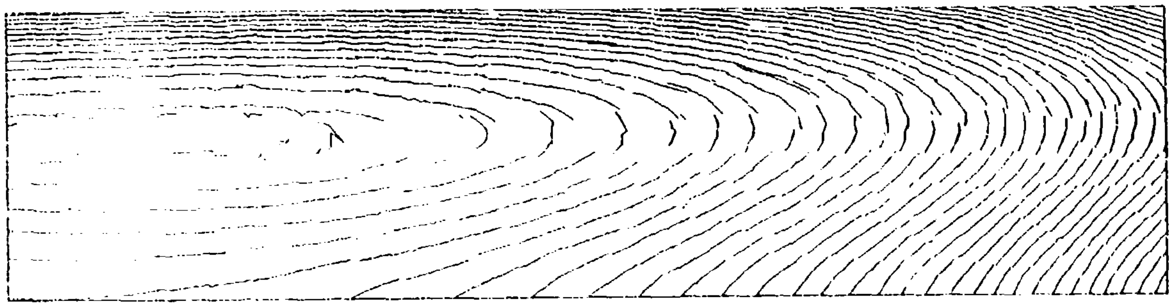


Fig. 29b: 2nd Order Reconstruction of P_3, Irregular Mesh (GC-HOR).

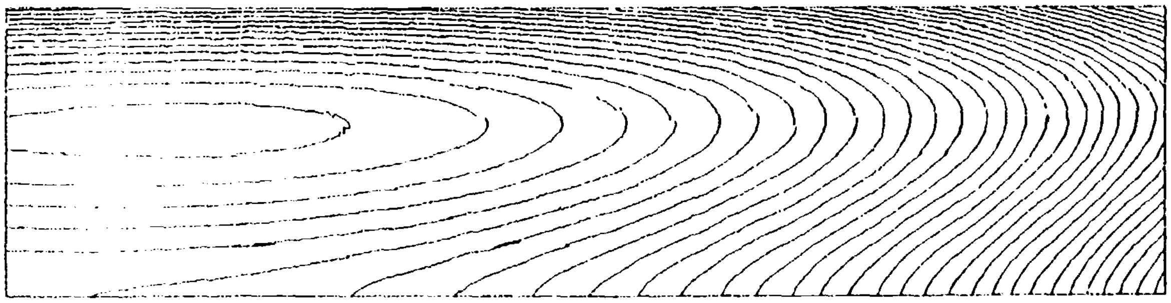


Fig. 29c: 3rd Order Reconstruction of P_3, Irregular Mesh (GC-HOR).

5 HOR-SCHEMES

This section discusses 2D upwind finite volume schemes applying higher order polynomial solution or flux reconstruction for linear and nonlinear convection problems with a source term. As yet, no attention will be paid to the monotonicity of the schemes nor will stability be taken into consideration.

5.1 Linear Problem

Consider the 2D linear convection equation with source term to be solved in a domain Ω:

$$\frac{\partial q}{\partial t} + \vec{a}.\vec{\nabla} q = S(q, \vec{r}) \qquad (97)$$

where $\vec{a}$ is a constant convection vector and q the solution of the analytical problem. Integration over Ω gives:

$$\iint_{\Omega} \frac{\partial q}{\partial t}.d\Omega + \iint_{\Omega} \vec{a}.\vec{\nabla} q.d\Omega = \iint_{\Omega} S(q, \vec{r}).d\Omega \qquad (98)$$

Discretizing the domain by partitioning it into smaller cells P of arbitrary polygonal shape and applying Green's theorem to the second term in the left hand side, eq. (98) can be written as:

$$\sum_P \iint_{\Omega_P} \frac{\partial q}{\partial t}.d\Omega + \sum_P \oint_{\partial\Omega_P} \vec{n}.\vec{a}.q.ds$$

$$= \sum_P \iint_{\Omega_P} S(q, \vec{r}).d\Omega \qquad (99)$$

whereby $\sum_P \Omega_P = \Omega$ and $\vec{n}$ is the outward unit normal vector at the edges of the cell P while the contour integration has to be carried out in counter-clockwise direction.

Eq. (98) will be solved exactly if the next equation is satisfied $\forall P \in \Omega$:

$$\iint_{\Omega_P} \frac{\partial q}{\partial t}.d\Omega + \oint_{\partial\Omega_P} \vec{n}.\vec{a}.q.ds = \iint_{\Omega_P} S(q, \vec{r}).d\Omega \quad (100)$$

Using the results of approximate Riemann solvers, an upwind space discretization of the left hand side of eq. (100) with a first order accurate time discretization is obtained and the scheme becomes (after division by Ω_P):

$$\frac{\overline{Q_P^{n+1}} - \overline{Q_P^n}}{\Delta t} + \left[D(\vec{a}.\vec{\nabla} q)\right](Q^n) = [D(S)](Q^n) \quad (101)$$

with

$$\left[D(\vec{a}.\vec{\nabla} q)\right](Q) = \frac{1}{\Omega_P}.\sum_R \sum_O \left(a_{PR}^+ .Q_{PO}^{(k)} + a_{PR}^- .Q_{RO}^{(k)}\right).\Delta s(102)$$

the space discretization, $D(S)$ an as yet undefined discretization of the source term S and

$$\begin{aligned} a_{PR}^+ &= \max(\vec{a}.\vec{n}, 0) \\ a_{PR}^- &= \min(\vec{a}.\vec{n}, 0) \end{aligned} \qquad (103)$$

from which it follows that

$$a_{PR}^+ + a_{PR}^- = \vec{a}.\vec{n} \ .$$

Remember that $\vec{a}$ is a constant vector. $\overline{Q_P^n}$ is the mean of the discrete solution distribution over the cell P at time level n.

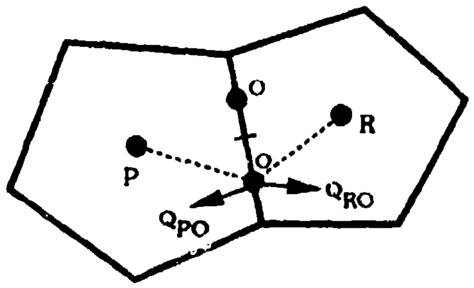


Fig. 30: Gauss points O on the edge between the cells P and R.

The value of $Q_{PO}^{(k)}$ and $Q_{RO}^{(k)}$ is the value of k^{th} order polynomial reconstruction at the Gauss points O on the cell interface between cell P and its neighbour R, seen from cell P resp. cell R (see also eq. (54) and fig. 30):

$$\begin{aligned} Q_{PO}^{(k)} &= \overline{Q_P^n} + \left(\mathbf{W}.\overline{\mathbf{Q}^n}\right)_P .\mathbf{F}^P(\vec{r}_O) && (104) \\ Q_{RO}^{(k)} &= \overline{Q_R^n} + \left(\mathbf{W}.\overline{\mathbf{Q}^n}\right)_R .\mathbf{F}^R(\vec{r}_O) && (105) \end{aligned}$$

or following eq. (77):

$$\begin{aligned} Q_{PO}^{(k)} &= Q_P^n + (\mathbf{W}.\mathbf{Q}^n)_P .\mathbf{G}^P(\vec{r}_O) && (106) \\ Q_{RO}^{(k)} &= Q_R^n + (\mathbf{W}.\mathbf{Q}^n)_R .\mathbf{G}^R(\vec{r}_O) && (107) \end{aligned}$$

in which $\mathbf{F}^P$ and $\mathbf{F}^R$ are the vectors containing the zero mean basis polynomials related to the reference points of cell P resp. R:

$$\begin{aligned} F^P_{i,j}(\vec{r}) &= (x-x_P)^i.(y-y_P)^j - I^{P,P}_{i,j} \\ F^R_{i,j}(\vec{r}) &= (x-x_R)^i.(y-y_R)^j - I^{R,R}_{i,j} \\ &\quad , \; 0 < i+j \leq k \end{aligned}$$

and where $\mathbf{G}^P$ and $\mathbf{G}^R$ are the vectors containing the basis polynomials for the GC-HOR-Algorithm related to the reference points of cell P resp. R:

$$\begin{aligned} G^P_{i,j}(\vec{r}) &= (x-x_P)^i.(y-y_P)^j \\ G^R_{i,j}(\vec{r}) &= (x-x_R)^i.(y-y_R)^j \\ &\quad , \; 0 < i+j \leq k \end{aligned}$$

See also eqs. (30), (33) and (76). Note also that the cell P and its direct neighbours R have in general a different support.

Theorem 5.1
The space discretization of the scheme given by eq. (101) and eq. (102) is order h^k accurate on any type of mesh wherever the solution is smooth:

$$\left[D\left(\vec{a}.\vec{\nabla}q\right)\right](q) = <\vec{a}.\vec{\nabla}q>^P + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q \qquad (108)$$

Note that in eq. (108) the symbol q denotes the solution of the exact solution of the (analytical) equivalent differential equation [30, 12], whereas in the following, the symbol Q stands for the discrete solution function at a given moment during the computation.
Two proofs for this theorem are given in appendix A. A first proof is based on eqs. (92) and (93) stating that the polynomial reconstruction approximates a solution $q(\vec{r})$ to order $h^{(k+1)}.\mathcal{D}^{k+1}_q$. The second proof uses a truncation error analysis based on the concept of the equivalent differential equation, see also [30, 12]. The choice of $D(S)$ will follow from each of the proofs, see subsection 5.4.

For the time being, no attention will be paid to the time discretization which nevertheless still has an effect on the space discretization as well.

5.1.1 Polynomial Preservation

Let us now investigate what happens to an exact polynomial solution q of degree r satisfying the steady state of eq. (97) in Ω when passing it into the scheme given by eq. (101). If the degree of the polynomial solution does not exceed the order of the reconstruction k, then $Q^{(k)}_{PO} = Q^{(k)}_{RO} = q_O$ and eq. (101) reduces to:

$$\frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} + \frac{1}{\Omega_P}.\sum_R\sum_O(\vec{a}.\vec{n}_{PR}).q_O.\Delta s_O = D(S)$$

If the number of Gauss points on each edge is larger than or equal to $\frac{r+1}{2}$, one finds, after applying the Gauss theorem:

$$\frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} + \frac{1}{\Omega_P}.\iint_{\Omega_P}\left(\vec{a}.\vec{\nabla}q\right).d\Omega = D(S)$$

As q satisfies $\vec{a}.\vec{\nabla}q = S(q,\vec{r})$ everywhere in Ω_P, this can be further reduced to:

$$\frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} + \frac{1}{\Omega_P}.\iint_{\Omega_P} S(q,\vec{r}).d\Omega = D(S)$$

or

$$\frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} = D(S) - <S(q,\vec{r})>^P \qquad (109)$$

The right hand side of eq. (109) will be zero if $D(S)$ is an exact discretization of cell average of the source term. This then means that the time change of the polynomial steady state solution is zero. In other words, an exact polynomial steady state solution of degree r is no longer modified by the scheme (101) if the the order of the solution reconstruction is at least equal to r.

Note that, in the general case, the difference between $D(S)$ and $<S(q,\vec{r})>^P$ is given by eq. (121) showing that the difference is not zero. It will be zero if the source term can be averaged exactly. This leads to the following theorem linking polynomial preservation with higher order space accuracy:

Theorem 5.2
If a scheme is polynomial preserving up to degree k and if its source term discretization is of order h^k in space, then the scheme is of order h^k in space.

Proof of Theorem 5.2:
Suppose that the scheme is lower order accurate:

$$D(\vec{a}.\vec{\nabla}q) = <\vec{a}.\vec{\nabla}q>^P + \mathcal{O}(h^p)\mathcal{D}^{p+1}_q$$

with $p < k$. Note also that, in the equivalent differential equation of a scheme, a derivative of order $p+1$ can never have a coefficient which is larger than something of order h^p. Hence, terms like $\mathcal{O}(h^p)\mathcal{D}^{k+1}_q$ with $p < k$ can never appear.

Plugging an exact polynomial solution q of degree k in the scheme of eq. (101) would lead to:

$$\begin{aligned} \frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} + \frac{1}{\Omega_P}.\iint_{\Omega_P}\left(\vec{a}.\vec{\nabla}q\right).d\Omega + \mathcal{O}(h^p)\mathcal{D}^{p+1}_q \\ = D(S) \end{aligned}$$

As it is supposed here that the source term is discretized with a space accuracy of $\mathcal{O}(h^k)$ with $k < p$, this becomes:

$$\frac{\overline{Q}^{n+1}_P - \overline{Q}^n_P}{\Delta t} = \mathcal{O}(h^p)\mathcal{D}^{p+1}_q \neq 0$$

This is in contradiction with the starting hypothesis that the scheme is polynomial preserving. So, polynomial preservation up to degree k implies k^{th} order space accuracy if the source term is discretized exactly.

□

5.2 Nonlinear Problem

Remind first that q stands for the exact solution of a given analytical problem whereas Q denotes the discrete solution at a certain moment during a computation.
Let the following 2D nonlinear convection equation with source term S to be solved in a domain Ω:

$$\frac{\partial q}{\partial t} + \vec{\nabla}.\vec{F}(q,\vec{r}) = S(q,\vec{r}) \qquad (110)$$

where $\vec{F}$ is some flux vector. Integration of eq. (110) over Ω gives:

$$\iint_\Omega \frac{\partial q}{\partial t}.d\Omega + \iint_\Omega \vec{\nabla}.\vec{F}(q,\vec{r}).d\Omega = \iint_\Omega S(q,\vec{r}).d\Omega \tag{111}$$

Proceeding as for the linear problem, eq. (111) can be written as:

$$\sum_P \iint_{\Omega_P} \frac{\partial q}{\partial t}.d\Omega + \sum_P \oint_{\partial\Omega_P} \vec{F}(q,\vec{r}).\vec{n}.ds = \sum_P \iint_{\Omega_P} S(q,\vec{r}).d\Omega \tag{112}$$

whereby $\sum_P \Omega_P = \Omega$ and $\vec{n}$ is the outward unit normal vector at the edges of the cell P while the contour integration has to be carried out in counter-clockwise direction. Eq. (111) will be solved exactly if the next equation is satisfied $\forall P \in \Omega$:

$$\iint_{\Omega_P} \frac{\partial q}{\partial t}.d\Omega + \oint_{\partial\Omega_P} \vec{F}(q,\vec{r}).\vec{n}.ds = \iint_{\Omega_P} S(q,\vec{r}).d\Omega \tag{113}$$

The flux function $H(q,\vec{r},\vec{n})$ is now defined as $\vec{F}(q,\vec{r}).\vec{n}$ and can be evaluated based on the concepts of 1D projected Riemann solvers leading to an upwind space discretization of the left hand side of eq. (113) with a first order accurate time discretization as given in the following equation (after division by Ω_P):

$$\frac{\overline{Q}_P^{n+1} - \overline{Q}_P^n}{\Delta t} + \left[D(\vec{\nabla}.\vec{F})\right](Q^n) = [D(S)](Q^n) \tag{114}$$

with

$$\left[D(\vec{\nabla}.\vec{F})\right](Q) = \frac{1}{\Omega_P}.\sum_R \sum_O \tilde{H}^O_{PR}.\Delta s_O \tag{115}$$

the space discretization, $D(S)$ an as yet undefined discretization of the source term S and $\tilde{H}_{PR}$ a numerical flux vector obtained using a 1D approximate Riemann solver. The latter can be either a flux vector splitter (FVS):

$$\tilde{H}^O_{PR} = H^+(Q^{(k)}_{PO},\vec{r}_O,\vec{n}_{PR}) + H^-(Q^{(k)}_{RO},\vec{r}_O,\vec{n}_{PR}) \tag{116}$$

or a flux difference splitter (Roe-type splitter, [19]):

$$\begin{aligned}\tilde{H}^O_{PR} = \; & \frac{1}{2}\left[H\left(Q^{(k)}_{PO},\vec{r}_O,\vec{n}_{PR}\right) + H\left(Q^{(k)}_{RO},\vec{r}_O,\vec{n}_{PR}\right)\right] \\ & - \frac{1}{2}|C_m(Q_m,\vec{n}_{PR})|.\left(Q^{(k)}_{RO} - Q^{(k)}_{PO}\right)\end{aligned} \tag{117}$$

where the value of $Q^{(k)}_{PO}$ and $Q^{(k)}_{RO}$ is the value of kth order polynomial reconstruction at the Gauss points O on the cell interface between cell P and its neighbour R, seen from cell P resp. cell R, see also eq. (104), (105), (106) and (107). The symbol C_m stands for the Jacobian $\frac{\partial H}{\partial Q}$ taken at the Roe-averaged state $Q_m\left(Q^{(k)}_{PO},Q^{(k)}_{RO}\right)$. Note also that for both the eqs. (116) and (117) the following is valid:

$$\tilde{H}_{PR}(Q,Q,\vec{r}) = H(Q,\vec{r},\vec{n}_{PR})$$

It will now be shown that the space discretization of the scheme given by eq. (114) and eq. (115) is order h^k accurate on any type of mesh for both flux vector and flux difference splitting schemes. Note that this will only be valid where H, H^+, H^- and Q are smooth functions. As for the linear problem, no attention will be paid to the time discretization which nevertheless still has an effect on the space discretization.

5.2.1 FVS-HOR-Scheme

Theorem 5.3
The space discretization of the scheme given by eq. (114) and eq. (116) is order h^k accurate on any type of mesh wherever the solution is smooth:

$$\left[D(\vec{\nabla}.\vec{F})\right](q) = <\vec{\nabla}.\vec{F}>^P + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q\mathcal{D}^1_H \tag{118}$$

Proof of Theorem 5.3:
To proof order h^k space accuracy, one studies the truncation error by deriving the equivalent differential equation using local Taylor series expansions. Using eqs. (92) or (93) and if the exact solution q of the equivalent differential equation varies smoothly, scheme (114) with (116) can then be rewritten as (when carrying out a Taylor series development of H^+_{PR} and H^-_{PR} around q_O):

$$\begin{aligned}\left[D(\vec{\nabla}.\vec{F})\right](q) = & \\ & \frac{1}{\Omega_P}.\sum_R\sum_O \left[H^+_{PR}(q_O) + \mathcal{O}(h^{k+1})\mathcal{D}^{k+1}_q\mathcal{D}^1_{H+}\right. \\ & \left. + H^-_{PR}(q_O) + \mathcal{O}(h^{k+1})\mathcal{D}^{k+1}_q\mathcal{D}^1_{H-}\right].\Delta s_O \\ = \; & \frac{1}{\Omega_P}.\sum_R\sum_O H(q_O,\vec{r}_O,\vec{n}_{PR}).\Delta s_O \\ & + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q\mathcal{D}^1_{H+,-} \\ = \; & \frac{1}{\Omega_P}.\sum_R\sum_O \left[H(q_P,\vec{r}_P,\vec{n}_{PR})\right. \\ & \left. + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}^P_l(\vec{r}_O).\partial\mathbf{H}^P_l\right].\Delta s_O \\ & + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q\mathcal{D}^1_H\end{aligned}$$

where the definition of $\mathbf{C}^P_l$ is given in eq. (90) while the one of $\partial\mathbf{H}^P_l$ is similar to the one found in eq. (69).

The previous expression for the space discretization can be rewritten as a continuous contour integral if the number of Gauss points O on the edges is larger than or equal to $\frac{k+1}{2}$:

$$\begin{aligned}\left[D(\vec{\nabla}.\vec{F})\right](q) = & \\ & \frac{1}{\Omega_P}.\oint_{\partial\Omega_P}\left[H(q,\vec{r},\vec{n}) + \mathcal{O}(h^{k+1})\right].ds \\ & + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q\mathcal{D}^1_H \\ = \; & <\vec{\nabla}.\vec{F}>^P + \mathcal{O}(h^k)\mathcal{D}^{k+1}_q\mathcal{D}^1_H\end{aligned}$$

which completes the proof of kth order space discretization. Note also that one is discretizing mean values over a cell even when the GC-HOR-Algorithm is used. Point values are discretized only if the order of the reconstruction $k \leq 1$.

□

5.2.2 FDS-HOR-Scheme

Theorem 5.4
The space discretization of the scheme given by eq. (114) and eq. (117) is order h^k accurate on any type of mesh wherever the solution is smooth:

$$\left[D(\vec{\nabla}.\vec{F})\right](q) = <\vec{\nabla}.\vec{F}>^P + \mathcal{O}(h^k)\mathcal{D}_q^{k+1}\mathcal{D}_H^1\mathcal{D}_A^1 \quad (119)$$

Proof of Theorem 5.4:
Again using eqs. (92) and (93) and if q varies smoothly, scheme (114) with (117) can be rewritten as (when carrying out a Taylor series development of H, q and C_m around q_O):

$$\begin{aligned}
D(\vec{\nabla}.\vec{F}) = &\\
&\frac{1}{2\Omega_P}.\sum_R\sum_O\Big\{\check{H}(q_O,\vec{r}_O,\vec{n}_{PR})\\
&+\mathcal{O}(h^{k+1})\mathcal{D}_q^{k+1}\mathcal{D}_H^1\\
&+\tilde{H}(q_O,\vec{r}_O,\vec{n}_{PR})+\mathcal{O}(h^{k+1})\mathcal{D}_q^{k+1}\mathcal{D}_H^1\\
&+\big[C_m(q_O,q_O)\\
&+\mathcal{O}(h^{k+1})\mathcal{D}_A^1\big]\mathcal{O}(h^{k+1})\mathcal{D}_q^{k+1}\Big\}.\Delta s_O\\
= \ &\frac{1}{\Omega_P}.\sum_R\sum_O H(q_O,\vec{r}_O,\vec{n}_{PR}).\Delta s_O\\
&+\mathcal{O}(h^k)\mathcal{D}_q^{k+1}\mathcal{D}_H^1\\
= \ &\frac{1}{2\Omega_P}.\sum_R\sum_O\left[H(q_O,\vec{r}_O,\vec{n}_{PR})\right].\Delta s_O\\
&+\mathcal{O}(h^k)\mathcal{D}_q^{k+1}\mathcal{D}_H^1\mathcal{D}_A^1
\end{aligned}$$

From here on, the same reasoning is followed as for the FVS-HOR-Scheme whereby it is required that the number of Gauss points O on the edges is larger than or equal to $\frac{k+1}{2}$. This brings one then to the conclusion that FDS-HOR-Schemes are k^{th} order space accurate on any type of mesh even if considerable grid irregularities are present. □

5.2.3 Polynomial Preservation

As with the linear problem, let us now investigate what happens to an exact polynomial solution q of degree r satisfying the steady state of eq. (110) in Ω when passing it in the scheme given by eq. (114). If the degree of the polynomial solution does not exceed the order of the reconstruction k, then $Q_{PO}^{(k)} = Q_{RO}^{(k)} = q_O$ and eq. (114) reduces to:

$$\begin{aligned}
\frac{\overline{Q}_P^{n+1}-\overline{Q}_P^{n}}{\Delta t} = &\\
&-\frac{1}{\Omega_P}.\sum_R\sum_O H(q_O,\vec{r}_O,\vec{n}_{PR}).\Delta s_O + D(S)\\
&-\frac{1}{\Omega_P}.\sum_R\sum_O\Big[H(q_O,\vec{r}_O,\vec{n}_{PR})\\
&+\sum_{l=1}^{r}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{H}_l^P\Big]\Delta s_O + D(S)\\
&+\mathcal{O}(h^r)\mathcal{D}_H^{r+1}
\end{aligned}$$

If the number of Gauss points on each edge is larger than or equal to $\frac{r+1}{2}$, one finds, after applying the Gauss theorem:

$$\frac{\overline{Q}_P^{n+1}-\overline{Q}_P^{n}}{\Delta t}+\frac{1}{\Omega_P}.\iint_{\Omega_P}\vec{\nabla}.\vec{F}.d\Omega = D(S)+\mathcal{O}(h^r)\mathcal{D}_H^{r+1}$$

from which it follows that nonlinear HOR-Schemes are only polynomial preserving if higher order derivatives of H w.r.t. to q are zero and if there is a sufficient number of Gauss points. As for the linear problem $D(S)$ should be an exact discretization of the source term. Note that, in the general case, the difference between $D(S)$ and $< S(q,\vec{r}) >^P$ is given by eq. (121) showing that the difference is not zero. It will be zero if the source term can be averaged exactly. In that case, it will be shown here that if a scheme is polynomial preserving up to degree k then it is of order h^k in space.

5.2.4 Flux Reconstruction

For nonlinear problems it seems logic to reconstruct the fluxes rather than the solution variables themselves, in particular when ENO-reconstruction is used, see section 6. It is expected to achieve a better control over the monotonicity of the scheme.
Shu and Osher [31, 14] devised indeed a Lax-Friedrichs type of scheme where the numerical flux

$$\tilde{F}(Q) = F(Q) + \alpha Q$$

is reconstructed using higher degree polynomials. They presented interesting results in two space dimensions.
A slightly different approach was brought up by Harten [16] whereby the averages of the reconstructed solution Q^{reco} and the reconstruction of the projected flux vector $\left(\vec{F}.\vec{n}_{PR}\right)^{reco} = H_{PR}^{reco}$ are evaluated over each cell edge using an appropriate Gauss quadrature formula. These edge-averaged values available from both sides of the edge are then used to solve a single Riemann problem using a Roe-type approximate Riemann solver rather than the number of times as would be required by the quadrature formula. This option depends of course strongly on the computational cost of the approximate Riemann solver.
Harten proposes two ways of reconstructing the fluxes. A first one consists of writing the physical flux vector as an analytical function of the solution reconstruction polynomial. This function is then rewritten as a kind of truncated Taylor expansion whereby analytical expressions for the space derivatives are used. The coefficients in this expansion are such that the reconstruction conserves the mean. A second method uses a gravity center reconstruction whereby the values of the physical flux vector at the cell centroids are evaluated using the value of the zero-mean solution reconstruction at those centroids. The first flux reconstruction method seems to be most appropriate for unstructured grid solvers implemented on parallel computers. The second method has a lower computational cost but requires more storage and results in more inter-processor communication.

5.3 Discussion

From eq. (108), (118) and (119) it follows that the schemes (101) and (114) can become inconsistent when piecewise constant solution reconstruction is used. In practice, piecewise constant solution reconstruction always gives reasonable results even on irregular meshes which indicates that the scheme is at least first order. LeVeque [32] showed in 1D that first order accuracy is indeed achieved on irregular meshes. The proof in 2D is presented for a linear scheme in appendix B and depends on the time discretization and the fact that the scheme is stable for CFL-numbers which are of order h.

Eq. (108), (118) and (119) indicate that the truncation error of the space discretization is at least of order h^k, which does not exclude that the complete scheme can be of order h^{k+1} not only on regular meshes but also on irregular meshes. Actually, the choice of the time discretization can be such that its truncation error compensates the terms of order h^k in the space discretization truncation error to order h^{k+1}. LeVeque made some suggestions in this direction, while Shu [31] used a Runge-Kutta type of explicit time stepping whereby the coefficients where optimized for space accuracy and stability. Further comments on time integration will be given in section 5.5.

Remark also that no lower order derivatives are present in the truncation error given by eq. (108), (118) and (119). In other words, terms of the form:

$$\mathcal{O}(h^r)\mathcal{D}_q^p \quad \text{with } r \le k \text{ and } p \le r$$

do not occur. This guarantees that the scheme has no second order diffusive errors if the order of the solution reconstruction is larger than 2. If the order of reconstruction is 1, the scheme might be second order accurate (see LeVeque) but still have diffusive errors. This implies the presence of the following term in the truncation error:

$$\mathcal{O}(h^2)\mathcal{D}_q^2 \quad \text{whereby } k = 1$$

This idea can be generalized to higher order stating that a scheme with an $\mathcal{O}(h^k)$ space operator might become $\mathcal{O}(h^{k+1})$ due to the time discretization while derivatives of order $k+1$ remain present in the truncation error which becomes then of the following shape:

$$\mathcal{O}(h^{k+1})\mathcal{D}_q^{k+1}$$

5.4 Source Term

Eq. (108), (118) and (118) indicate that the discretization of the source term $S(q,\vec{r})$ must satisfy the following equation when $\mathcal{O}(h^k)$ space accuracy is to be obtained:

$$[D(S)](q) = < S(q,\vec{r}) >^P + \mathcal{O}(h^p) \text{ with } p \ge k \quad (120)$$

If S is a known function of $\vec{r}$ and linear in q then $D(S)$ is taken as the exact mean of the function over the cell P with q taken in the gravity center. In all other cases the following discretization is proposed:

$$[D(S)](Q) = S\left[Q_P^{(k)}(\vec{r}_P),\vec{r}_P\right] + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{I}_l^{P,P}.\partial\mathbf{S}_l^{*,P} \quad (121)$$

in which

$$S^*(\vec{r}) = S\left[Q_P^{(k)}(\vec{r}),\vec{r}\right] \quad (122)$$

and where the vector $\partial\mathbf{S}_l^{*,P}$ is defined in a similar way as in eq. (69). The derivatives in this vector take known expressions. The following theorem is going to be proved:

Theorem 5.5
The source term discretization given by eq. (121) satisfies the condition (120) for higher order space accuracy.

Proof of Theorem 5.5:
One assumes that the exact averages of a smooth exact solution q are given in each cell P. If now also S is a smooth function of q then from eq. (122) and eq. (92) and after performing a Taylor series development of S around q it follows that:

$$S^*(\vec{r}) = S(q,\vec{r}) + \mathcal{O}(h^{k+1})\mathcal{D}_S^q \quad (123)$$

This leads now to the following relation between higher order derivatives of S and S^*:

$$\partial\mathbf{S}_l^{*,P} = \partial\mathbf{S}_l^P + \mathcal{O}(h^{(k+1)-l})\mathcal{D}_S^q \quad (124)$$

Inserting eq. (123) and (124) in eq. (121), one finds:

$$\begin{aligned}[D(S)](q) &= S\left[q(\vec{r}_P),\vec{r}_P\right] \\ &\quad + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{I}_l^{P,P}.\partial\mathbf{S}_l^P + \mathcal{O}(h^{k+1}) \\ &= < S(q,\vec{r}) >^P + \mathcal{O}(h^{k+1}) \end{aligned} \quad (125)$$

□

Remark that a simpler source term discretization can be used in combination with the GC-HOR-Algorithm:

$$\begin{aligned}[D(S)](Q) &= S(Q_P,\vec{r}_P) \\ &+ \sum_{i+j>0}^{i+j\le k} I_{i,j}^{P,P}.\sum_{R^*} W_{i,j}^{R^*,P}.S(Q_{R^*},\vec{r}_{R^*})\end{aligned} \quad (126)$$

i.e. the reconstruction of the source term using the point values at the reference points of the cells. The symbol Q still denotes a discrete solution at a certain moment during the computation. The discretization (126) satisfies also condition (120) as long as S and the exact solution q are smooth functions. To proof this, it is sufficient to let $D(S)$ act upon the exact solution q and to perform a Taylor series development of $S(q_{R^*},\vec{r}_{R^*})$ around the point $\vec{r}_P$ and truncate after the terms of order k.

5.5 Time Discretization

The problem at hand is to find a good approximation for the time dependent term

$$\frac{1}{\Omega_P}\iint_{\Omega_P}\frac{\partial q}{\partial t}.d\Omega \equiv < q_t >^P \quad (127)$$

in eq. (100) and eq. (113) where q is the exact solution of the respective analytical problems. The discretization is fairly different for ZM-HOR and GC-HOR-Algorithms algorithms described in section 4. Therefore, the time discretization for schemes based on the ZM-HOR and the GC-HOR-Algorithm will be discussed separately for the linear problem with source term given by eq. (97). Other time discretizations reported in the literature will be addressed as well.

5.5.1 ZM-HOR-Algorithm

Consider the following discretization $D(q_t)$ of the term given in eq. (127):

$$[D(q_t)](Q) = \frac{\overline{Q_P}^{n+1} - \overline{Q_P}^n}{\Delta t} \quad (128)$$

where Q is a discrete solution, see also eq. (101). The mean values in the right hand side of eq. (128) could be computed as the mean of the reconstruction polynomial over cell P. However, from eq. (54) and the fact that zero-mean polynomials are used, it follows that the cell average of the k^{th} order reconstruction of Q equals $\overline{Q}_P$. Hence, no cell averaging phase is needed in the updating phase of the computation. From eq. (101) one finds then that the new cell value of the discrete solution at the next time step can immediately be evaluated as:

$$\overline{Q}_P^{n+1} = \overline{Q}_P^n - \Delta t.\left[D(\vec{a}.\vec{\nabla}q) + D(S)\right](Q^n) \qquad (129)$$

Let us now investigate the truncation error of the discretization given in eq. (128) by inserting the exact solution q and performing a Taylor expansion in time and space around $\overline{q}_P^n \equiv \, < q(\vec{r},t^n) >^P$. One gets:

$$\begin{aligned}[D(q_t)](q) &= \frac{< q(\vec{r},t^{n+1}) >^P - < q(\vec{r},t^n) >^P}{\Delta t} \\ &= \, < q_t >^P + \mathcal{O}(\Delta t) \qquad (130)\end{aligned}$$

5.5.2 GC-HOR-Algorithm

The time discretization in eq. (101) or eq. (128) is based on mean values of the solution over the cell P, also for the scheme using the GC-HOR-algorithm. However, unlike the scheme using the ZM-HOR-algorithm, the mean of the solution over the cell is not stored but the value of the solution at some reference point (in general the gravity center). Hence, a averaging phase will be needed in the updating phase of the computation:

$$\begin{aligned}\overline{Q}_P^{n+1} - \overline{Q}_P^n &= \, < Q_P^{n+1} + \mathbf{W}.\mathbf{Q}^{n+1}.\mathbf{G} >^P \\ &\quad - < Q_P^n + \mathbf{W}.\mathbf{Q}^n.\mathbf{G} >^P \\ &= \Delta Q_P + \mathbf{W}.\Delta\mathbf{Q}.\mathbf{I}^{P,P} \qquad (131)\end{aligned}$$

where $\mathbf{W}$, $\mathbf{G}$ and $\mathbf{Q}$ are defined in eq. (58), (78) resp. (80), vector $\mathbf{I}_{P,P}$ is given by:

$$\mathbf{I}^{P,P} = \left[I_{1,0}^{P,P} \cdots I_{i,j}^{P,P} \cdots I_{0,k}^{P,P}\right]^T \qquad (132)$$

and ΔQ_P and $\Delta\mathbf{Q}$ stand for the time changes of the solution values in centroids of cell P and resp. all cells belonging to the support of cell P. This means that eq. (131) leads to a system of equations since it has to be satisfied for all cells simultaneously. This system could be solved with any direct or iterative inversion algorithm. Since this is an expensive operation, mass lumping can be applied just as is done in Finite Element techniques. This means that instead of averaging the complete reconstruction, only the constant part Q_P is taken into account. However, since

$$< Q_P^{(k)} >^P = Q_P + \mathcal{O}(h).\mathcal{D}_Q$$

the averaging phase will introduce an additional first order space error. If one is only interested in steady state solutions whereby $\mathcal{D}_Q^t$ tends to zero, this temporarily increased truncation error is acceptable. The new discrete values at a next time level are then given by:

$$Q_P^{n+1} = Q_P^n + \Delta t\left[D(S) - D(\vec{a}.\vec{\nabla}q)\right](Q^n)\,,\ \forall P \quad (133)$$

5.5.3 Runge-Kutta Methods

The general explicit p-stage Runge-Kutta method can be written as:

$$Q^{(i)} = Q^{(0)} + \Delta t\sum_{k=0}^{i-1} c_{ik}\mathcal{L}\left[Q^{(k)}\right] \qquad (134)$$

with

$$\mathcal{L}\left[Q^{(k)}\right] = \left[D\left(\vec{a}.\vec{\nabla}_q\right) + D(S)\right](Q^{(k)})$$

whereby $i = 1, 2, ..., p$ and $Q^{(0)} = Q^n$ and $Q^{(p)} = Q^{n+1}$. The integer p denotes the total number of intermediate steps.

Shu and Osher [31] have chosen the coefficients c_{ik} such that the scheme is TVD under certain CFL-restrictions (a detailed discussion of stability and Total Variation is deferred to section 6). They came up with second, third, fourth and fifth order schemes of which only the second order scheme is presented here:

$$\begin{aligned}Q^{(1)} &= Q^{(0)} + \Delta t\mathcal{L}\left[Q^{(0)}\right] \\ Q^{(2)} &= Q^{(0)} + \frac{1}{2}\Delta t\left\{\mathcal{L}\left[Q^{(0)}\right] + \mathcal{L}\left[Q^{(1)}\right]\right\} \qquad (135) \\ CFL &\le 1 \qquad (136) \\ & \qquad (137)\end{aligned}$$

Note that the CFL-restriction is valid only in 1D computations. For two calculations the limit is 1/2. Note that ENO-reconstruction is needed after each intermediate time step, see section 6.

Harten [16] proposed a single step higher order accurate time stepping scheme avoiding multiple invocations of ENO-reconstruction algorithms. Harten's approach is based on a Cauchy-Kowalewski procedure whereby the solution $Q(\vec{r},t)$ is written in a Taylor series expansion in both space and time whereby:

$$\begin{aligned}Q_{xt} &= S_x - a.Q_{xx} - b.Q_{xy} \\ Q_{yt} &= S_y - a.Q_{xy} - b.Q_{yy} \\ Q_{tt} &= S_t - a.Q_{xt} - b.Q_{yt} \\ &= -a.S_x - b.S_y + a^2.Q_{xx} + 2ab.Q_{xy} + b^2.Q_{yy} \\ &= -\left[\left(\vec{a}.\vec{\nabla}\right)S\right] + \left[\left(\vec{a}.\vec{\nabla}\right)^2 Q\right] \\ &\ \ \vdots\end{aligned}$$

These expressions are of course only valid if one solves the linear convection problem given by eq. (97). Note also that one can then show via a recursive proof that:

$$\frac{\partial^p Q}{\partial t^p} = (-1)^{p-1}.\left[\left(\vec{a}.\vec{\nabla}\right)^{p-1}S\right] + (-1)^p.\left[\left(\vec{a}.\vec{\nabla}\right)^p Q\right]$$

It is clear that the Taylor series expansion of the reconstructed solution $Q(\vec{r},t)$ taken at the initial time level t^n will be equal to the reconstruction polynomial at this time level. Hence, all mixed time/space derivatives can be expressed in pure spatial derivatives which can be evaluated using the reconstruction polynomials. The scheme becomes then:

$$\begin{aligned}\overline{Q}_P^{n+1} &= \overline{Q}_P^n \\ &\quad - \int_{t^n}^{t^{n+1}}\left\{\left[D(\vec{a}.\vec{\nabla}q) + D(S)\right](Q(\vec{r},\tau))\right\}.d\tau\end{aligned}$$

The continuous time integration is then approximated by some Gauss quadrature satisfying the desired time accuracy. The values of the solution at intermediate time levels and at the several Gauss points on the cell interfaces are evaluated using the Taylor series expansion in both time and space. The CFL-limit for this scheme is about 1/2.

5.6 Distinction with Finite Elements

HOR-Schemes differ from Finite Element methods (see the book of Johnson C. [33]) in that their data representation is allowed to be discontinuous whereas classical Finite Elements methods, such as the Taylor-Galerkin or the Streamwise Upwind Petrov-Galerkin (SUPG) methods [34, 10], use a continuous data interpolation based on Galerkin basis functions. The discontinuous interpolation used in the HOR-Schemes allows a straightforward incorporation of one dimensional Riemann solvers having upwind properties. Note, however, that recently Finite Element discretizations have been devised using discontinuous solution representations.

Finally, the spatial size of the support increases with the order of accuracy in HOR-Schemes. The order of a Finite Element scheme is increased by adding new degrees of freedom in the same element hence avoiding a spatial growth of the support.

6 ADAPTIVE STENCILS

6.1 Introduction

A general explicit scheme can be cast as, see also eqs. (101,(102) and (114),(115):

$$Q_P^{n+1} = D(Q_{R^*}^n) \qquad (138)$$

with D some discretization operator acting upon the solution given in each cell R^* of the support. The scheme (138) is called Total Variation Diminishing (TVD) if:

$$TV\left(Q^{n+1}\right) = TV\left[D\left(Q^n\right)\right] \leq TV\left(Q^n\right) \qquad (139)$$

with

$$TV\left(Q^n\right) = \sum_P \sum_{R>P} |Q_R^n - Q_P^n| \qquad (140)$$

in which R points to a direct neighbour of cell P. Harten [35, 36] showed the following properties of linear schemes for scalar problems; i.e. the operator D is linear:

1. A monotone scheme (D is a monotone function of its arguments) is always TVD, but not vice versa.
2. All coefficients in a monotone scheme are positive.
3. A TVD-scheme is always monotonicity preserving, but not vice versa. Monotonicity preservation implies that if Q is monotone on a given mesh, then $D(Q)$ will also be monotone on the same mesh.
4. A monotonicity preserving scheme is always at most first order accurate in space. From this it follows that monotone and TVD-schemes are also always first order accurate.

For nonlinear schemes, however, the following properties are valid:

1. A monotone scheme is always at most first order accurate in space.
2. All coefficients in a monotone scheme are positive.
3. A TVD-scheme is always monotonicity preserving, but not vice versa.
4. A monotonicity preserving scheme can be higher order accurate in space, hence also a TVD-scheme.
5. If one has a uniform boundedness of the TV then convergence to a weak solution is achieved.
6. If the TV of the discrete solution at time level $n+1$ does not exceed the TV of the reconstruction of the solution at time level n or, in other words, if $TV\left[D\left(Q^n\right)\right] \leq TV\left(Q^{reco,n}\right)$, then the scheme is TVD.
7. TVD-schemes are always stable but are at most second accurate in space.

Considering the schemes given by eq. (101) and (114) with the above properties in mind, one finds that they are not monotone nor TVD. Hence, convergence is not at all guaranteed. In order to stabilize these higher order schemes, their discretization operator D must be made nonlinear.

As higher order accuracy is required, the resulting nonlinear schemes cannot be TVD since the latter can not achieve more than second order accuracy. One tries therefore to make an "almost TVD" scheme based on Essentially Non-Oscillatory (ENO) solution reconstruction; i.e. a reconstruction satisfying:

$$TV\left(Q^{reco}\right) \leq TV(Q) + \mathcal{O}\left(h^{k+1}\right) \qquad (141)$$

As the support of the reconstruction is arbitrary, the condition (141) can be fulfilled by selecting the support in an adaptive way such that no solution discontinuities of order less than or equal to k are traversing the zone of the support. The nonlinearity of the discretization operator lies then in the fact that cells are accepted or rejected from the support after each iteration. Such a scheme is also called a moving stencil scheme (MS). As the order of accuracy is independent on the support choice (see section 5), the accuracy of MS-schemes is not affected by moving the stencils. Hence, MS-schemes cannot be TVD. However, no proof is available yet that would guarantee the uniform boundedness of the TV and hence convergence. Nevertheless, the numerous computations reported on in the literature indicate the stable properties of MS-schemes.

In this section, two adaptive support selection algorithms will be discussed. The first algorithm performs a support gathering by heuristic marching (SSM-Algorithm, Support Selection by Marching). The second algorithm selects the support out of a set of candidate supports such that the global norm of the derivatives of the reconstruction polynomial are minimal at the gravity center (SSG-Algorithm, Support Selection for Global norm minimization).

6.2 SSM-Algorithm

6.2.1 General Concept

The support of a cell P always contains the cell P itself whereafter the two direct neighbours making the smallest gradient are added. The gradient associated with two direct neighbours is estimated using a local triangulation connecting the gravity centers of the direct neighbours with the centroid of cell P, see fig. 31.

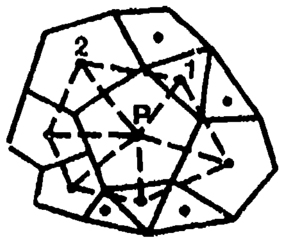


Fig. 31: Local triangulation for gradient estimation.

One assigns the given cell averages (ZM-HOR-Algorithm) or point values (GC-HOR-Algorithm) of the solution function q to the vertices of each local triangle. The gradient in such triangle is then obtained through a linear interpolation over the triangle:

$$\vec{\nabla} q\Big|_{\Delta P12} = \frac{1}{2\Omega_{\Delta P12}} \cdot \begin{bmatrix} \Delta q_{P1}\Delta y_{P2} - \Delta q_{P2}\Delta y_{P1} \\ \Delta q_{P1}\Delta x_{P2} - \Delta q_{P2}\Delta x_{P1} \end{bmatrix} \tag{142}$$

where

$$\Delta q_{P1} = q_1 - q_P \quad \text{and} \quad \Delta q_{P2} = q_2 - q_P$$

and similar definitions for the coordinates x and y.

After inserting cell 1 and 2 in the support, a marching procedure is started from cell 1. Another pair of direct neighbours of cell 1 is added if the gradient differs least from the gradient that was computed when cell 1 was added to the support. In other words, the support is selected in the zone where the solution function q remains as close as possible to a 3D plane passing through the point $\vec{r}_P(x_P, y_P, q_P)$ and having $\vec{\nabla} q_{1,P}$ as a gradient.

The selection algorithm includes central biasing. This means that a central support will be selected if the gradients estimated with each pair of direct neighbours differ very little. In that case all direct neighbours are included. Central biasing is interesting since the round-off error for the reconstruction is then minimal.

Finally, note also that the algorithm breaks down on regular meshes if the order of the reconstruction is larger than one. Problems as described in section 4.5 occur in this case.

6.2.2 Detailed Procedure

Note that the selection procedure is implemented in an Object-Oriented manner using C++. Therefore the following discussion will be more data and event related instead of tracing the more traditional paths of flow charts and functionalities.

6.2.3 Startup

The minimum number of cells n needed in the support is set equal to:

$$n = \frac{1}{2}(k+1)(k+2) + 2$$

Two cells are added in order to have better conditioned matrices, in particular for regular meshes (see section 4.5). Besides this, a discontinuity sensor ϕ is initialized to 5 (see subsection 6.2.5).

The support used during the previous iteration is emptied and the current cell P is added as a first cell in the support for the iteration at hand.

6.2.4 Search for Lowest Gradient

Using the local triangulation with the neighbours of P as described in subsection 6.2.1, one stores the corresponding norms of the gradients in increasing order. The value following which the neighbours are sorted is called the sorting variable which now is the square of the norm of the gradients of the variable q one is reconstructing. The associated first and second neighbour of each triangle are listed in the same order as well as the gradient vectors. The object containing all this information is called a neighbour sorter, named NS_P:

$\|\vec{\nabla}\|^2$	N_1	N_2	$\vec{\nabla}$
10	2	3	(3,1)
20	1	2	(2,4)
106	3	1	(9,5)

where N_1 and N_2 are the lists of the first resp. second neighbour forming the local triangle.

For each neighbour R of P another neighbour sorter NS_R is generated whereby the sorting variable is given by the following expression:

$$\frac{\left\|\vec{\nabla} q_{i,R} - \vec{\nabla} q_{R,P}\right\|^2}{\left\|\vec{\nabla} q_{R,P}\right\|^2} \tag{143}$$

in which the first subscript i is associated with the local triangle $\Delta_{R,i,i+1}$ as depicted in fig. 32, while the second subscript P or R denotes the cell around which the local triangulation is made.

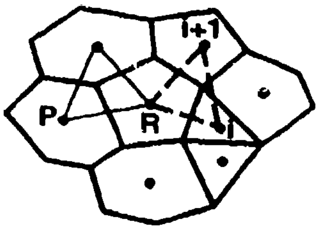


Fig. 32: Triangle $\Delta_{R,i,i+1}$.

The lowest sorting variable value of each generated neighbour sorter is multiplied with the square of the norm of the gradient in the associated triangle $\Delta_{R,i,i+1}$:

$$\left(\min_i \frac{\left\|\vec{\nabla} q_{i,R} - \vec{\nabla} q_{R,P}\right\|^2}{\left\|\vec{\nabla} q_{R,P}\right\|^2}\right) \cdot \left\|\vec{\nabla} q_{i,R,\min}\right\|^2 \tag{144}$$

This value is now used as the sorting variable for a new neighbour sorter NS_{PR} acting upon the direct neighbours R of the cell P. In other words, a neighbour R comes first in the neighbour sorter if its smallest relative gradient difference is small as well as the gradient with which the smallest relative gradient was achieved.

6.2.5 Discontinuity Detection

The procedure described in the previous subsection results in a list of neighbours ordered with increasing associated gradient norms and norms of gradient differences. This is still not yet sufficient to come to the final decision of accepting a neighbour into the support. Indeed, the norm of the gradient differences might be smaller across a discontinuity. Hence, a separate discontinuity detection is required.

Given the gradient vectors stored in the neighbour sorter NS_P, the discontinuity sensor ϕ is defined as:

$$\phi = \frac{\sum_{R=1}^{N} \left\| \vec{\nabla} q_{R,P} - \vec{\nabla} q_{R,P,min} \right\|^2}{N . \left\| \vec{\nabla} q_{R,P,min} \right\|^2} \qquad (145)$$

where $\left\| \vec{\nabla} q_{R,P,min} \right\|$ is the minimum norm of the gradient vectors given in the sorter NS_P while N is the number of direct neighbours of the cell P. The sensor ϕ is a measure of the gradient variation around the cell P. A cell R belonging to the list N_1 of the sorter NS_P will be added to a list of rejected cells if:

$$\phi > THRES \ (= 0.1 \text{ initially}) \qquad (146)$$

and

$$\min \left(\left\| \vec{\nabla} q_{R,P} \right\|^2 , \left\| \vec{\nabla} q_{R-1,P} \right\|^2 \right) > \left\| \vec{\nabla} q_{R,P,min} \right\|^2 . \left(1 + \frac{\phi}{5} \right) \qquad (147)$$

where the subscript R points now to the R^{th} cell in the cyclic list giving the direct neighbours of the cell P in counter-clockwise direction, see also fig. 33.

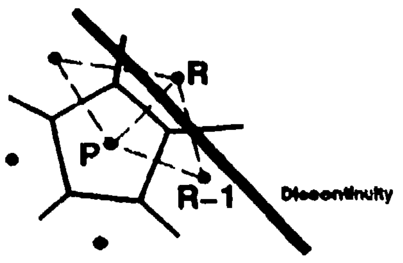


Fig. 33: Rejecting cells at a discontinuity.

The coefficient 5 in the denumerator in the right hand side of eq. (147) was determined empirically to comply with the widest possible range of gradient values.

6.2.6 Acceptance of Support Cells

Denoting the sorting variable value of neighbour R in the neighbour sorter NS_{PR} by SV_R, a cell in the list N_1 and the associated cell in the list N_2 forming the local triangle will be admitted to the support if:

- the number of cells n already present in the support satisfies:

$$n \leq \frac{1}{2}(k+1)(k+2) + 2 \qquad (148)$$

-
$$\frac{SV_R - SV_{R,min}}{SV_{R,max} - SV_{R,min}} < \epsilon \qquad (149)$$

  where $0 < \epsilon < 1$

- the candidate cell is not in the list of rejected cells and not already present in the support.

The second condition leads to a central stencil if the gradient is almost constant, i.e. for almost linearly varying solution functions. The parameter ϵ is called the central bias parameter: the higher its value is set, the more central supports will be obtained. Cells out of the list N_2 are added simultaneously to the support in order to avoid having a one dimensional support leading to the problem discussed in section 4.5.

When a cell of the list N_1 is accepted, a branching flag is set at the next cell in the list N_1. This is relevant to the marching procedure described in subsection 6.2.7.

6.2.7 Marching

Looping now over the new cells added to the support, the whole procedure starting from subsection 6.2.4 is repeated. Again, one loops over the latest added cells and so on.

If one is in a region enclosed by discontinuities or domain boundaries, it could be that no more cells can be admitted. In this case, one tries to make a branch in the marching at cells where a branching flag was set. If this does not lead to additional support cells, one loops down in the support and tries again to add other neighbours. If still this remains unsatisfactory, the threshold for the discontinuity sensor ϕ is doubled and the procedure starts all over again. This can be repeated until the threshold becomes so high that the weakest discontinuities are just seen as sharp gradients. The reconstruction will then no longer be truly ENO and the only cure is the have more and finer cells in these zones.

6.2.8 Boundaries

Arriving in a boundary cell, fig. 34, one sorts first the neighbours of the interior neighbour R based on the norms of the absolute gradient vectors, see section 6.2.4.

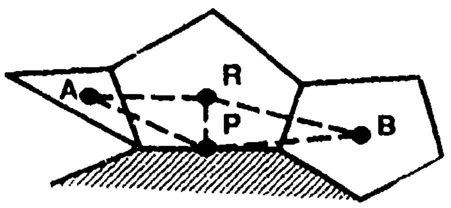


Fig. 34: Gradient estimation at boundaries.

The cells A and B are direct neighbours of the interior neighbour R which are vertices of the two local triangles that have also the boundary cell P as a vertex. Their neighbours are sorted using the following sorting variable:

$$SV_A = \left\| \vec{\nabla} q_{i,A} - \vec{\nabla} q_{A,P} \right\|^2$$
$$SV_B = \left\| \vec{\nabla} q_{i,B} - \vec{\nabla} q_{B,P} \right\|^2$$

The neighbours of the interior neighbour are once again sorted but now using the relative norm of the difference of the respective gradient vectors with the mean of the gradient vectors. The result is stored in a neighbour sorter NS_R. The lists N_1 and N_2 of the sorter NS_R contain each both the cells A and B. If $SV_A < SV_B$, one selects cell A from list N_1 together with the corresponding sorting variable value and gradient vector in NS_R. These three items are then stored in the neighbour sorter NS_{PR} of the boundary cell P. If $SVA \geq SV_B$, the cell B is selected from the list N_2 of NS_R as well as the corresponding sorting variable value and gradient vector. Again, these items are stored in NS_{PR}.

The neighbour sorter NS_{PR} finally contains a degenerate triangle $\Delta_{R,P,R}$ while the gradient vector is the gradient in the local triangle $\Delta_{R,A,P}$ or $\Delta_{R,P,B}$ having the smallest relative difference in gradient vectors. The sorting variable value is the square of the relative norm of the difference between the gradient vector in the selected triangle and the mean gradient vector of the interior neighbour R.

The reason behind this rather complex procedure is the fact that at weak discontinuities either cell A or B lies on the other side will still having the lowest absolute gradient vector. This is anticipated by looking to the smallest relative gradient vector differences of both cell A and B. The result might be that one has to select the neighbour with the higher gradient but with the smallest gradient variation.

6.3 SSG-Algorithm

The SSG-Algorithm is a generalization towards meshes with arbitrary polygonal cells of a similar algorithm proposed by Harten [16] for triangular grids. The algorithm suggests itself a set of candidate supports around a cell P and selects then the support for which the global norm of the derivatives of the reconstruction polynomial at the gravity center of the cell P is minimal. The global norm $\|D\|_P^2$ is given by:

$$\|D\|_P^2 = \sum_{i+j>0}^{i+j\leq k} \left(\frac{\partial^{i+j} q^{reco}}{\partial x^i \partial y^j}\bigg|_P \right)^2 \qquad (150)$$

whereby

$$\frac{\partial^{i+j} q^{reco}}{\partial x^i \partial y^j}\bigg|_P = i!\,j!\,A_{i,j} \qquad (151)$$

for both the GC-Algorithm and the ZM-Algorithm. The coefficient $A_{i,j}$ is an element of the vector $\mathbf{A}$ given by for instance eq. (37). This implies that the weight matrix $\mathbf{W}$ is to be recalculated for each candidate support.

6.3.1 Candidate Supports

The first candidate is always the central support, i.e. the support obtained by taking the neighbours of the neighbours and so on without taking any other limitation into account. After this, N sector supports are gathered with N the number of sides of a cell P. The gravity center of the cells of a sector support lies entirely in a sector limited by the two lines connecting the gravity center of cell P with two consecutive vertices, see fig. 35.

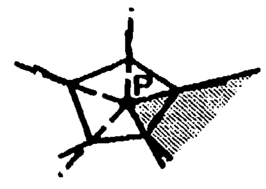


Fig. 35: Division by Sectors.

Note that a point P lies within a sector defined by two half lines L_1 and L_2 defined by their direction vector $\vec{a}_1$ resp. $\vec{a}_2$ if, see fig. 36:

$$\vec{a}_1 \otimes \vec{r}_{AP} \geq 0 \quad \text{and} \quad \vec{a}_2 \otimes \vec{r}_{AP} < 0 \qquad (152)$$

where the vector $\vec{r}_{AP}$ connects the position of the apex A of the sector with the position of the point P.

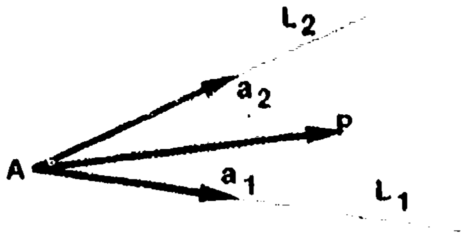


Fig. 36: Condition to ly within a sector.

The conditions to be satisfied in order to accept a cell in a sector support are the same as for the SSM-Algorithm except for condition (149).
For each of the $N+1$ candidate supports the global norm $\|D\|_P^2$ is computed and the candidate with the smallest global norm is selected. Note also that the norm of the central candidate support is divided by two in order to establish a bias towards central supports in order to improve the precision, see also Harten and Chakravarthy [16] or Shu [37].

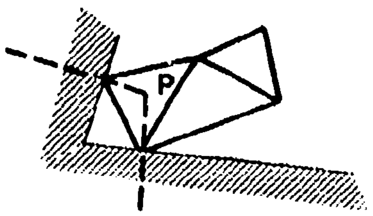


Fig. 37: Too small sector supports in corners.

6.3.2 Exceptions

1. It might happen in corners formed by two discontinuities or by two domain boundaries that not enough cells can be accepted, see fig. 37.

   The opening angle of the sector is then gradually increased until enough cells can be accepted. Candidate supports will then of course be partially overlapping. If still not enough cells can be found, the opening angle of the sector is decreased back to its initial value but the value of $THRES$ in the shock detection condition (146) is increased by a factor 5 until enough cells were gathered.

   If no cells were ever rejected due to condition (146) and if a large number of sector widenings took place, the candidate is no longer considered when still not enough support cells were found. In this case, one was dealing with a corner of the domain where the solution variables were varying smoothly.

2. When a neighbour of cell P is a boundary cell, the associated sector support is always too small and hence is not considered as a candidate support, see fig. 38.

6.3.3 Boundaries

The candidate supports for the reconstruction in boundary cell P consist of the central support of the interior neighbour and of the sector supports associated with the interior neighbour. However, the latter do not contain

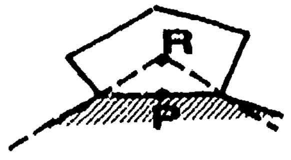


Fig. 38: Sector formed with a boundary cell.

the interior neighbour itself. This is done in order to enhance the chances for ENO-reconstruction when a discontinuity is present in between the gravity centers of the boundary cell and its interior neighbour.

Note that except for the central supports, no boundary cells are ever allowed in the support since having a boundary cell in the support close to a discontinuity may lead to instabilities. Indeed, a boundary cell which was in a smooth region may suddenly be in a non-smooth zone after applying the boundary conditions which lead to a change in the solution value stored in the boundary cell. When calculating afterwards the flux balances in the interior domain, the reconstruction in cells close to the boundary will no longer be ENO; hence, large oscillations will occur. This could eventually be cured by selecting new supports after updating the boundary cells but this would be far too costly.

6.4 Examples

In the following, some reconstructions of known polynomials with a discontinuity are presented here. Only examples using the GC-HOR-algorithm combined with the SSM-Algorithm are given as similar results were obtained with the ZM-HOR-algorithm.
The next set of functions were used to demonstrate the monotonicity of the reconstruction:

$$\left|\begin{array}{lcl} P_1(x,y) & = & 2x + 3y \\ P_2(x,y) & = & 3x^2 + 5y^2 + 2xy \\ & & \quad + 2x + y \\ P_3(x,y) & = & 3x^3 + 5y^3 + (x+y)^2 \\ P_4(x,y) & = & x^3y - 2y^4 - 2xy^3 \\ & & \quad + x + y \end{array}\right. \tag{153}$$

if $y > \tan(30^\circ).(x-1)$ and

$$\left|\begin{array}{lcl} P_1(x,y) & = & 2x - 3y - 5 \\ P_2(x,y) & = & -5x^2 + 6y^2 - 2xy \\ & & \quad - x + y + 5 \\ P_3(x,y) & = & 6(x-3)^3 - 5y^3 \\ & & \quad - (x+y)^2 + 10 \\ P_4(x,y) & = & -8(x-3)^3y + 4y^4 + 2xy^3 \\ & & \quad - 2x - y - 30 \end{array}\right. \tag{154}$$

otherwise. Only some results for functions P_3 and P_4 will be presented as analogue figures were obtained for the linear and quadratic functions. Fig. 39 shows a quadratic reconstruction of the cubic function P_3 whereby this function was also used to detect discontinuities.
Fig. 40 presents the support for a number of cells close to the discontinuity and in the smooth region. It is observed that the supports tend to align themselves with the isolines.

Finally, fig. 41 gives the isolines 4th order ENO-reconstruction of function P_4 demonstrating that both polynomials on either sides of the discontinuity can be reconstructed exactly (P_4 was also the function to detect discontinuities).
Only the shape of the discontinuity is not exact. One sees that it follows exactly the grid lines instead of being a straight line. There are methods to cure this problem like for instance the subcell resolution technique proposed by Harten et al. (ref. [38]), see also section 7.3

In the following, the reconstruction of the exact solution of the following problem is discussed:

$$q_t + q_x + \left(\frac{q^2}{2}\right)_y = 0 \tag{155}$$

with

$$\begin{aligned} q &= +1 \quad \text{on} \quad y = 0 \\ q &= -1 \quad \text{on} \quad y = +1 \\ q &= -2y+1 \quad \text{on} \quad x = 0 \ . \end{aligned}$$

Equation 155 refers to a convection problem whose convection speed is given by $(1, q)$. The solution is equal to -1 on the lower half of the domain and equal to +1 on the upper half. In the triangular zone in the left half the solution is given by:

$$q(x,y) = \frac{2y-1}{2x-1} \tag{156}$$

and varies linearly along lines where $x = etc$. At the borders of the triangular zone one observes therefore a discontinuity in the first derivatives of q and a discontinuity along the line $y = 0.5$ if $x > 0.5$. Fig. 42 shows the isolines of the quadratic reconstruction of the exact solution q using the GC-HOR-Algorithm combined with the SSG-Algorithm. The discontinuity along $y = 0.5$ is not visible. For reasons of comparison, fig. 43 shows the isolines obtained when performing a linear interpolation between the gravity centers of the grid. The discontinuity takes now the thickness of one cell. Fig. 44 shows a 3D view of the solution over the two dimensional domain whereby each "tile" in the plot represents the reconstruction polynomial in each grid cell. The discontinuity is infinitely sharp but again follows the grid lines.
Using the SSM-Algorithm in order to obtain an ENO-reconstruction, the supports are allowed to cross the discontinuity in the first derivative at the border of the triangular zone. Indeed, when marching, one takes first the neighbours forming the smallest first order derivative. In 1D, this can lead to the situation given in fig. 45.

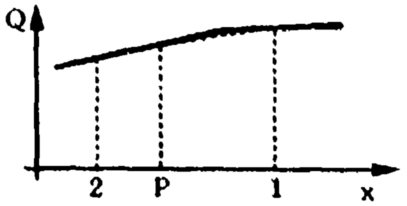


Fig. 45: Discontinuous derivatives in 1D.

It is apparent that first neighbour 1 of cell P will be taken. The marching procedure will then continue on the side with the smallest gradient and as cell P also belongs to the support, the reconstruction will oscillate close to cell P. Marching support selection algorithms should therefore be used with care.

Other two dimensional support selection algorithms are given in the literature. Abgrall [39] describes an algorithm for triangular meshes which is similar to the SSM-

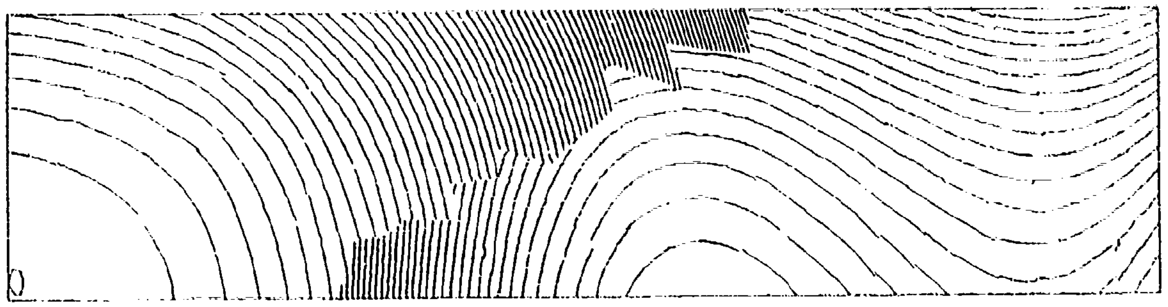


Fig. 39: **Isolines of function P_3 with quadratic ENO-reconstruction.**

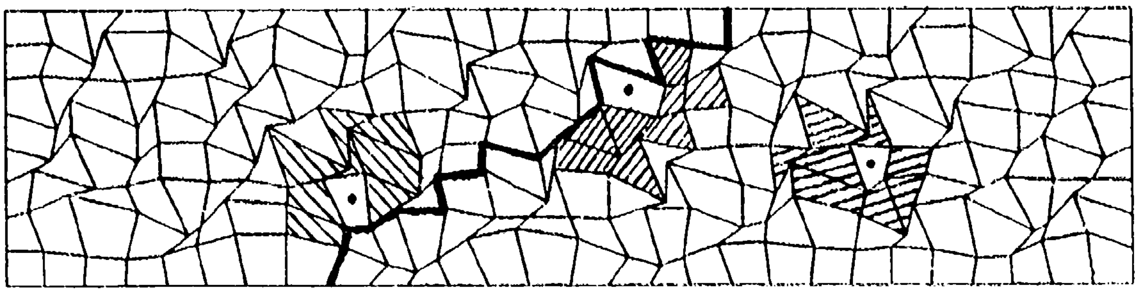


Fig. 40: Some supports for quadratic ENO-reconstruction of function P_3.

Algorithm. Harten [16] presents another marching algorithm which mathematically minimizes the higher order derivatives. He also proposes a fixed central support scheme for problems without discontinuities. This choice is justified by the fact that central supports are most accurate when reconstructing. An automatic switch between constant solution reconstruction and higher order reconstruction is introduced when solving problems with weak shocks (hybrid reconstruction). This will of course not lead to uniformly higher order schemes.
Shu and Osher [14] report on the effect of the choice of the first neighbouring support cell on the stability. They developed a support selection algorithm which takes the first neighbour support cell always on the upwind side.
Finally, it is sometimes suggested to keep the supports fixed in smooth zones in order to reduce the computational cost. However, it still remains to verify that such schemes are stable since the solution can start to oscillate also in smooth zones because the scheme is not TVD. This means that the supports will have to be adapted which increases again the computational cost.

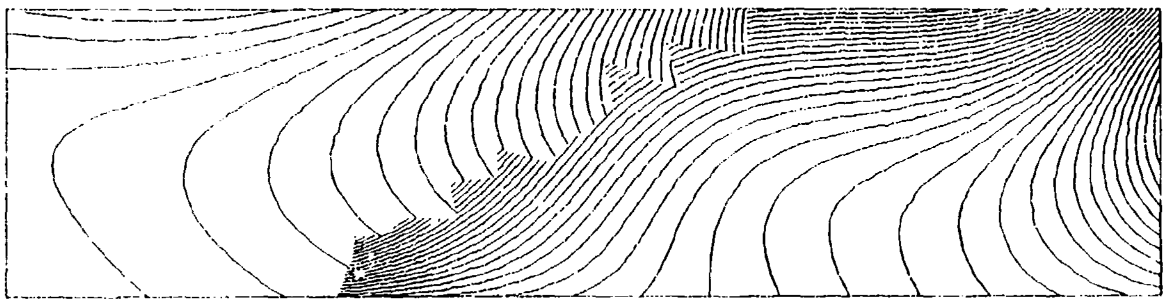


Fig. 41: Isolines of 4th order ENO-reconstruction of function P_4.

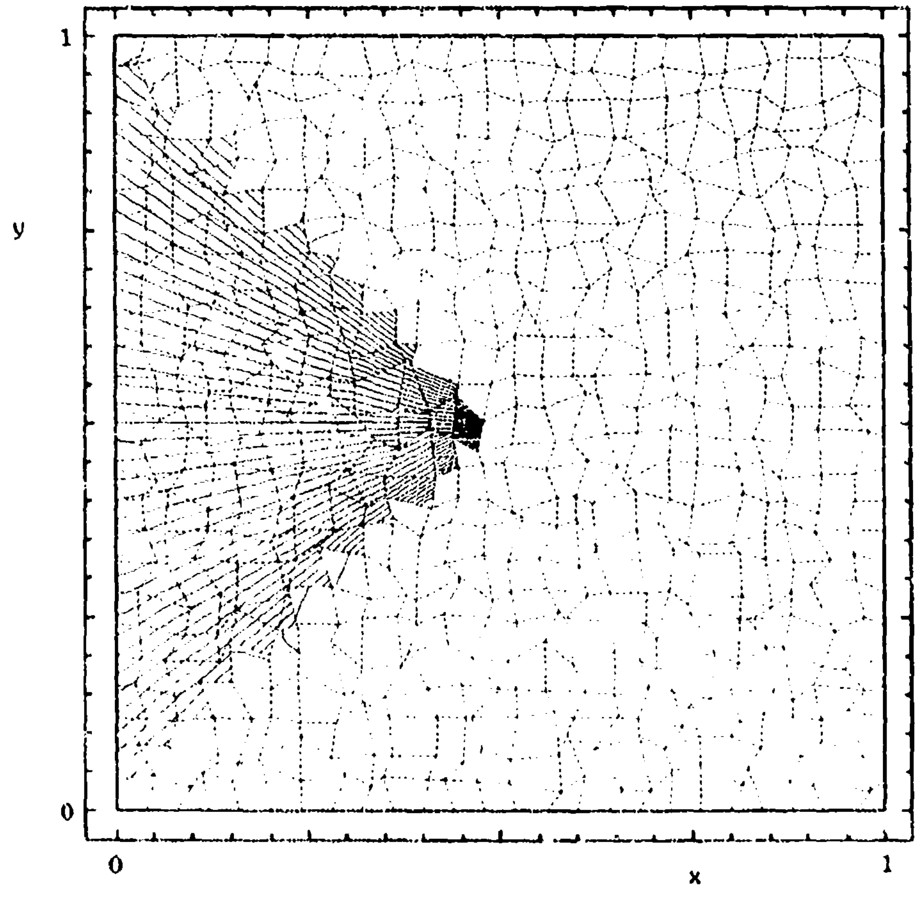


Fig. 42: Quadratic reconstruction of the exact solution of the nonlinear scalar convection problem (GC-HOR + SSG).

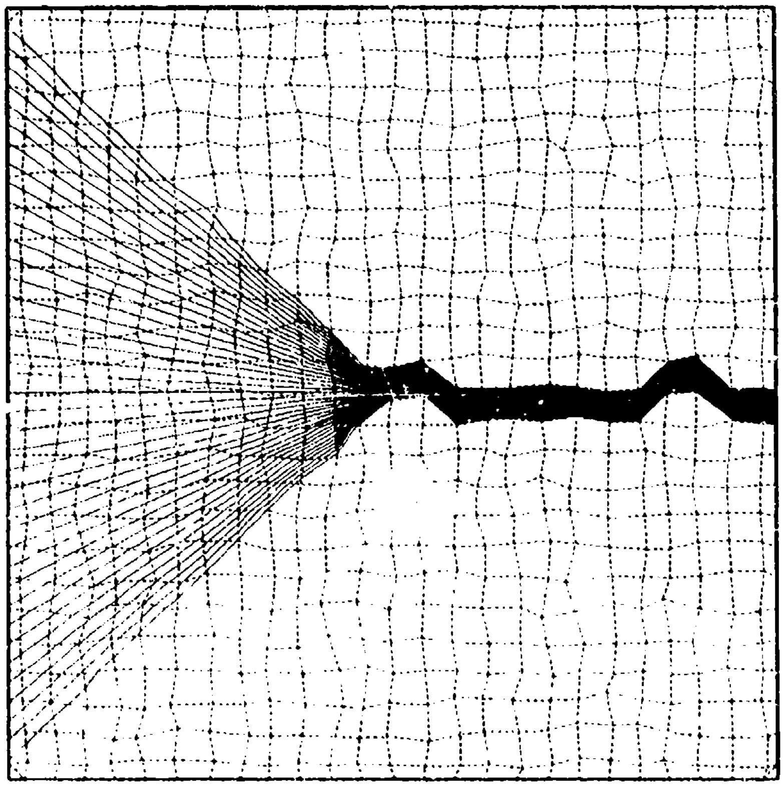


Fig. 43: Linear interpolation isolines of the exact solution of the nonlinear scalar convection problem.

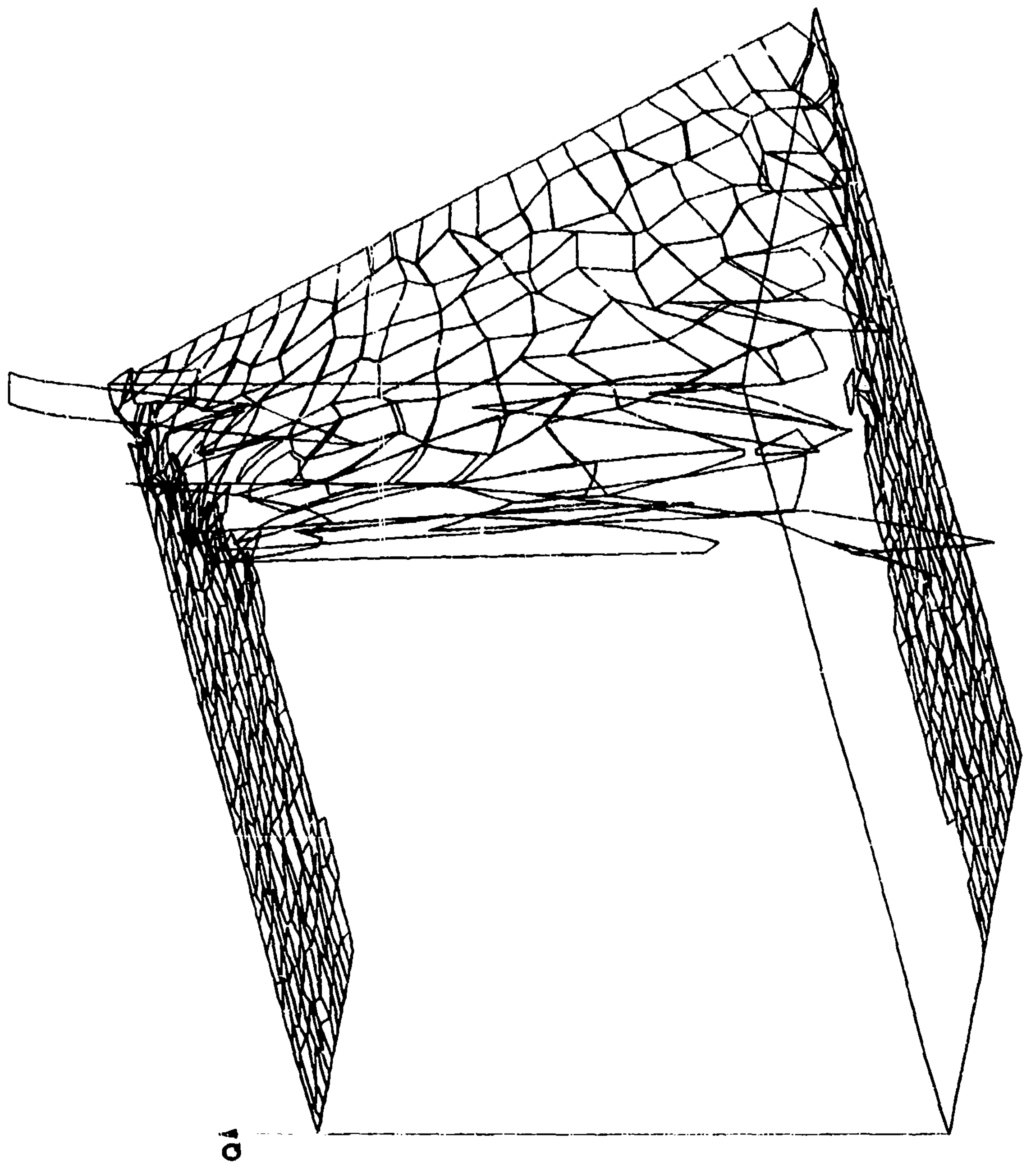


Fig. 44: Nonlinear scalar convection problem, 3D view of the quadratic reconstruction of the exact solution (GC-HOR + SSG).

7 SYSTEMS OF EQUATIONS

7.1 Introduction

Until now only scalar problems were tackled but the final aim is to solve a system of equations such as the Euler equations. Although the extension towards systems of equations is rather straightforward, some additional topics are particularly interesting when solving the Euler equations (2),(3),(4). This section will treat the reconstruction phase when solving systems of equations and also some recently developed concepts for sharpening discontinuities.

7.2 Reconstruction

In the case of the two dimensional Euler equations, one is now dealing with a solution vector Q of 4 solution variables. Two approaches are now possible [16], the component-wise support selection and the vector support selection.

The component-wise support selection assigns a support to each component of the solution vector. It increases the work spent in the support selection algorithm by a factor 4 but seems to be more robust when several moving shocks intersect in time-dependent problems. Note that in this case there is not always enough room to find a support in which no discontinuities occur and hence spurious oscillations can be generated.
The vector support selection assigns one and the same support to all 4 solution variables. The selection is performed on basis of the smoothness of a single (derived) variable such as the Mach number, static pressure, density,... This approach is considerably more economic.

Concerning the robustness of a scheme to be used for the Euler equations it is important to remind that the density ρ, static pressure p and static temperature T are nonnegative quantities. Reconstruction of the conservative variables in an ENO-fashion gives a direct control over the density but not over the derived quantities p and T. Small oscillations in the reconstruction of the conservative variable close to multiple discontinuity interactions lead in general to larger oscillations in p and T. When computing these quantities at a cell interface using the local values of the reconstruction of the conservative variables, one can get a negative value as a consequence of which the solution procedure is to be stopped.
As the characteristic variables vary more smoothly than the conservative variables at discontinuity interactions, it is often suggested in the literature to perform a local transformation to the characteristic variables. One can then compute all derived variables from the component-wise reconstruction of the characteristic variables which remain smooth at discontinuity interactions. The transformation matrix contains the eigenvectors of the Jacobian matrix of the flux normal to the discontinuity.
Another approach is to reconstruct the vector $(\rho, \rho u, \rho v, p)^T$. This gives again direct control over the density and the pressure and moreover, the normal momentum and p remain smooth across contact discontinuities.

7.3 Subcell Resolution

In 1D, Harten [38] proposed a method to resolve discontinuities more sharply. A discontinuity is detected by:

1. inspecting the norms $\sigma_i, \sigma_{i+1}, \sigma_{i-1}$ of the derivatives of the reconstruction in cell i and its neighbouring cells $i+1$ and $i-1$, fig. 46.
2. comparing the mean of the reconstruction polynomials Q_{i-1}^{reco} and Q_{i+1}^{reco} assigned to the neighbours $i-1$ resp. $i+1$ with the given discrete mean value $< Q >_i$ in cell P

A discontinuity is assumed when:

$$\begin{aligned} \sigma_i &> \sigma_{i-1} \\ \sigma_i &> \sigma_{i+1} \end{aligned} \qquad (157)$$

and

$$\left(< Q_{i+1}^{reco} >^i - < Q >^i\right).\left(< Q_{i-1}^{reco} >^i - < Q >^i\right) \leq 0 \qquad (158)$$

in which $< Q_{i+1}^{reco} >^i$ and $< Q_{i-1}^{reco} >^i$ denote the mean of the reconstruction polynomial of cell $i+1$ resp. $i-1$ over cell i. The reconstruction in cell i will then consist of the extension of the reconstruction polynomials of the

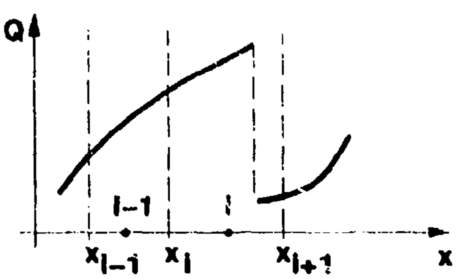


Fig. 46: 1D discretization around a discontinuity.

neighbours into the cell i, fig. 47. The interface between the two polynomials is located at a position x_j such that:

$$x_j(\theta) = x_i + \theta(x_{i+1} - x_i) \tag{159}$$

with θ the solution of the next nonlinear equation stating that the mean in the cell i must be conserved:

$$< Q_{i-1}^{reco} >^{i,j} + < Q_{i+1}^{reco} >^{j,i+1} = < Q >_i \tag{160}$$

in which $< Q_{i-1}^{reco} >^{i,j}$ and $< Q_{i+1}^{reco} >^{j,i+1}$ indicate the mean of the reconstruction polynomial of cell $i-1$ over the interval $[i,j]$ resp. the mean of the reconstruction polynomial of cell $i+1$ over the interval $[j, i+1]$.
Knowing θ, a scheme can be constructed while introducing a corrective flux [38].

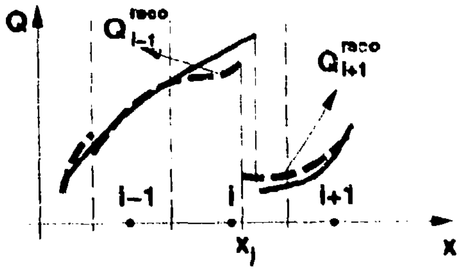


Fig. 47: 1D reconstruction with subcell resolution.

A 2D extension might consist of detecting the discontinuities based on the conditions (146) and whereby the normal direction of the discontinuity at a cell P is indicated by the mean gradient vector over all local triangles around P, fig. 31 and fig. 48. The polynomials of the two opposite direct neighbours having their centroid closest to the normal through the centroid of P will be extended into cell P. The interface line between the two polynomials is parallel to the discontinuity and its location is computed while requiring that the mean over the cell P be conserved.

7.4 Slope Modification Method

Yang [40] proposed to add a modification to the coefficients of the linear terms in the reconstruction polynomial. This modification is called a slope modifier and is a minmod-like function depending on the jumps in the unmodified reconstruction polynomials at the cell interfaces. The final effect is that one added artificial compression to the scheme.
Yang produced impressive results for both 1D and 2D problems. Moreover, his method is less costly than the subcell resolution approach. For a detailed comparison between the subcell and the slope modification approach, the reader is referred to the work of Shu and Osher [14].

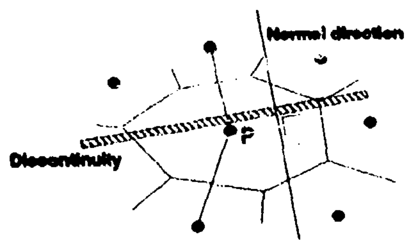


Fig. 48: Subcell resolution in 2D.

8 NUMERICAL EXPERIMENTS

8.1 Introduction

Results will be presented for linear scalar problems, nonlinear scalar problems and an Euler flow problem. All results were obtained with an object-oriented code written in C++ in order to have an enhanced flexibility when testing and adding new algorithms. Comparison of the code with an existing FORTRAN code when using exactly the same first order scheme on the same mesh did not reveal any loss of efficiency due to the object-oriented implementation.

8.2 Linear Scalar Problems

In this section, the verification of polynomial preservation will be addressed. Consider now the following problem on a rectangular domain Ω, see also fig. 26c:

$$\vec{a}.\vec{\nabla}q = S(x,y) \text{ with } \vec{a} = (2,1) \tag{161}$$

This problem was studied with 4 different source terms S given by:

$$\begin{aligned}
S_1(x,y) &= x + 2y - 3 \\
S_2(x,y) &= x + 2y - 3 \\
S_3(x,y) &= \frac{3}{8}x^2 + 33y^2 - \frac{1}{19}y \\
S_4(x,y) &= 8x^3 + 80y^3 + 24xy^2 - 68
\end{aligned}$$

The exact solution to the problem (161) is then given by following 4 polynomials:

$$\begin{aligned}
P_1(x,y) &= 2x + 3y + 2 \\
P_2(x,y) &= \frac{1}{4}x^2 + xy \\
&\quad - x - y + \frac{5}{4} \\
P_3(x,y) &= \frac{1}{76}x^3 + 11y^3 + \frac{1}{16}x^2 \\
&\quad - \frac{19}{2}y^2 - \frac{1}{4}xy + 10 \\
P_4(x,y) &= x^4 + 16y^4 + 8xy^3 \\
&\quad - 34x + 10
\end{aligned}$$

which are the same polynomials as those in eq. (94) of section 4. These solutions can of course only be obtained while imposing proper boundary conditions.

Tests were performed using both the GC-HOR and ZM-HOR-Schemes. In the following, one will discuss how polynomial preservation was achieved for the two algorithms.

For the GC-HOR-scheme, the exact solution q is given in the gravity centers of each cell :

$$Q_P = q(x_{G_P}, y_{G_P})$$

while for the ZM-HOR-scheme it is given as its exact mean value over each cell:

$$<Q>^P = \iint_{\Omega_P} q(x,y) \,.\, d\Omega\,,$$

for both interior and boundary cells.

It was then numerically observed that the imposed exact polynomial solution q of degree r was not touched, i.e. the residuals are within the round-off error of the machine (10^{-13}), if

- the exact flux integrals are imposed at the boundaries where the convection vector $\vec{a}$ is pointing into the domain.
- the same treatment as for interior edges is applied at boundaries where the convection vector points out of the domain. This is correct as the value in the boundary is not used when calculating the flux. Indeed, the upwind approximate Riemann solver will only take the value in the interior neighbour into account.
- the source term must be averaged exactly over each cell, see also eq. (120):

$$D(S) = \iint_{\Omega_P} S(x,y).d\Omega$$

- the order of the reconstruction k is at least equal to the degree r of the exact polynomial solution
- the number of Gauss points n satisfies the next condition:

$$n \leq \frac{k+1}{2}$$

  where k is the order of the reconstruction.

Following theorem 5.2, this experiment can be regarded as a numerical verification of the accuracy in the interior domain of the GC-HOR and ZM-HOR-scheme.

Fig. 49 shows the quadratic reconstruction of the numerical solution obtained using a GC-SSG-scheme with quadratic reconstruction, a 2 point Gauss quadrature (GP=2) along the edges and the source term S_4. The linear interpolation of the same solution is presented in fig. 50 which is to be compared with the exact solution shown in fig. 25. Note that using quadratic reconstruction for a polynomial solution of degree 4 does not preserve the solution since the order of reconstruction does not match the degree of the solution. The grid one used is a randomized rectangular mesh containing 296 cells, fig. 26c. The randomizing of the grid points is necessary to avoid a contradictory system for the reconstruction weights as discussed in subsection 4.5.
The solution was extracted after 100 iterations and took 90 minutes of CPU on an SGI Iris 4D/35 workstation, the final residual is of the order 10^{-3}.

Remark that another HOR-scheme can be devised whereby the solution in each cell is represented by only the linear part of the higher degree (k) reconstruction polynomial. Polynomial preservation for such a LINEX scheme (LINear EXtrapolation) is then obtained if:

- the order of the reconstruction is at least equal to the degree r of the polynomial exact solution
- only the exact solution value is imposed at the gravity centers of the boundary cells and not the exact flux.
- the source term is discretized as follows:

$$S(x_{G_P}, y_{G_P}) \,.\, \Omega_P$$

- only one (1!) Gauss point is used
- a non-conservative flux distribution is applied. This means that each edge is visited twice but each time seen from the other side. Thus:

$$\begin{aligned} Q_P^{n+1} &= Q_P^n - F_{PR} \\ Q_R^{n+1} &= Q_R^n - F_{RP} \end{aligned}$$

  whereby

$$\begin{aligned} F_{PR} &= \sum_R \vec{a}.\vec{n}_{PR}.Q_P^+ \\ F_{RP} &= \sum_P \vec{a}.\vec{n}_{RP}.Q_R^- \end{aligned}$$

  in which Q_P^+ and Q_R^- are the extrapolated values at the cell edges while using only the linear terms of the reconstruction polynomial. It is clear that the fluxes sent to the two cells adjacent to the same edge will not be identical so that the scheme is not conservative.
- at boundaries where the convection vector points outward the domain, the update of the boundary cell P is done as follows:

$$Q_P^{n+1} = Q_R^{(k)}(\vec{r}_P)$$

  where the superscript (k) indicates that all terms of the reconstruction polynomial in the interior neighbour R are taken into account. For the computation of the flux contribution to the interior neighbour still only the linear terms are used.

The LINEX scheme can become economic for reconstructions of order higher than or equal to 4 since then normally one would need at least 3 flux evaluations per edge instead of 2 in the case of the LINEX scheme. Also the evaluation of the source term is cheaper as one just has to take its value at the gravity center (even if ZM-reconstruction is used). However, as we are interested in capturing discontinuities correctly, the LINEX scheme is of few importance. Note that other non-conservative schemes with reduced grid sensitivity are described in the literature, see e.g. [9].

8.3 Nonlinear Scalar Problems

In this section, the numerical solution of the nonlinear problem of eq. (155) in section 6 will be discussed. The definition of the problem is repeated here:

$$q_t + q_x + \left(\frac{q^2}{2}\right)_y = 0 \qquad (162)$$

with

$$\begin{aligned} q &= +1 \quad \text{on} \quad y = 0 \\ q &= -1 \quad \text{on} \quad y = 1 \\ q &= -2y+1 \quad \text{on} \quad x = 0 \ . \end{aligned}$$

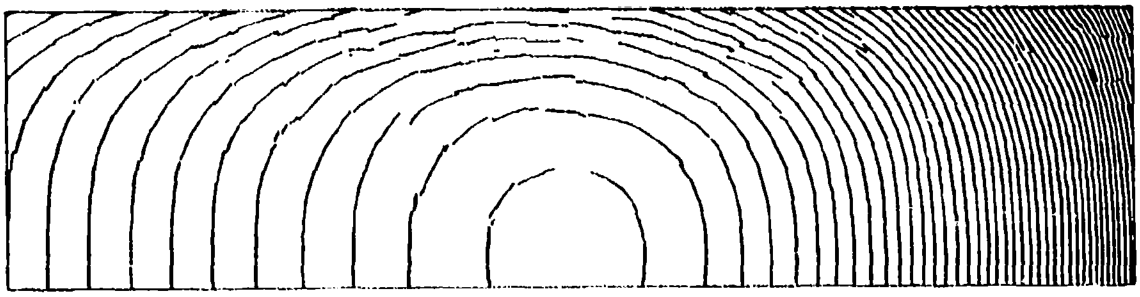


Fig. 49: Reconstruction of the solution of the linear problem, (k=2, GP=2, GC-SSG).

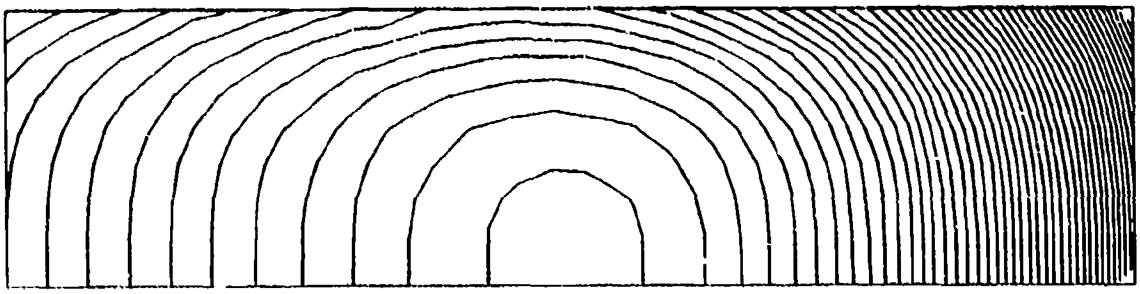


Fig. 50: Linear interpolation of the solution of the linear problem, (k=2, GP=2, GC-SSG).

The exact solu.ic is given in fig. 42 and 43. The computation started uom the exact solution using the GC-HOR reconstruction (k=2) with SSG support selection. The analytic flux is given by:

$$H = q.n_x + \frac{q^2}{2}.n_y \qquad (163)$$

The numerical fluxes $\bar{H}_{PO}$ through the edges were computed using a Roe-type flux splitter whereby C_m in eq. (8) is now given by:

$$C_m = n_{x_{PR}} + \frac{Q_{PO}^{(k)} + Q_{RO}^{(k)}}{2}.n_{y_{PR}} \qquad (164)$$

A two point Gauss quadrature (GP=2) was applied, meaning that the Riemann solver was invoked twice for each edge.

The exact solution was kept fixed at the upper, lower and left boundaries of the computational domain while the reconstructed solution in the interior cells at the right boundary was copied into the boundary.

The quadratic reconstruction of the numerical solution as well -- the mesh is shown in fig. 51 while the linear interpolation of the numerical solution on the dual mesh is presented in fig. 52. These two figures can be compared with fig. 42 resp. 43. A three dimensional view of the reconstruction of the obtained numerical solution is given in fig. 53 which can be compared with fig. 44.

It is seen that the discontinuity is captured in one cell and that no oscillations occur. This is put in evidence in fig. 54 and 55 showing a cut of the solution at $x = 0.3$ resp. $x = 0.95$.

The solution shown here was obtained after 100 Euler explicit time steps with a CFL-number equal to 0.1. The convergence history is given in fig. 56 showing the RMS-value over the whole domain of the relative change of the solution values over each time step. The total CPU cost of the calculation was about 3 hours on an SGI Iris 4D/35 workstation which means about 0.16 seconds per cell per iteration.

8.4 Ringleb Flow in a Bended Channel

The results presented in this subsection were computed using a fixed stencil and are to be considered as preliminar in particular with respect to the boundary condition treatment.

8.4.1 Problem Description

The Ringleb flow [41] is basically a potential flow in a strongly bended duct and can be described analytically. A proper choice of the shape of the duct guarantees that the flow remains everywhere subsonic, see fig. 57 taken from [42].

At any point in the domain, the flow is characterized by two parameters $\bar{q}$ and k. The parameter $\bar{q}$ is a non-dimensionalized speed and is constant along circles which therefore become the isotach lines. The parameter k is equal to $\frac{\bar{q}}{\sin\theta}$ where θ is the angle of the local flow direction. One can proof that k is a constant along streamlines.

Taking now the streamline with $k = 0.8$ as the inner wall of the bended duct will result in a fully subsonic flow. The outer wall coincides with the streamline for which $k = 0.4$. The in- and outlet of the duct are part of the isotach

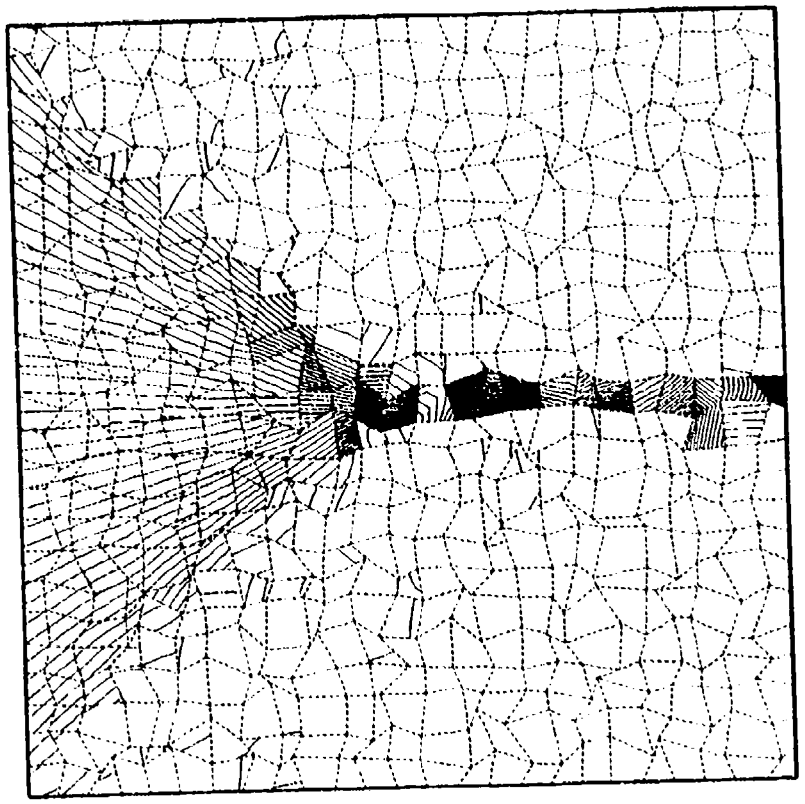


Fig. 51: Reconstruction of the computed solution of the nonlinear scalar convection problem, (k=2, GP=2, GC-SSG).

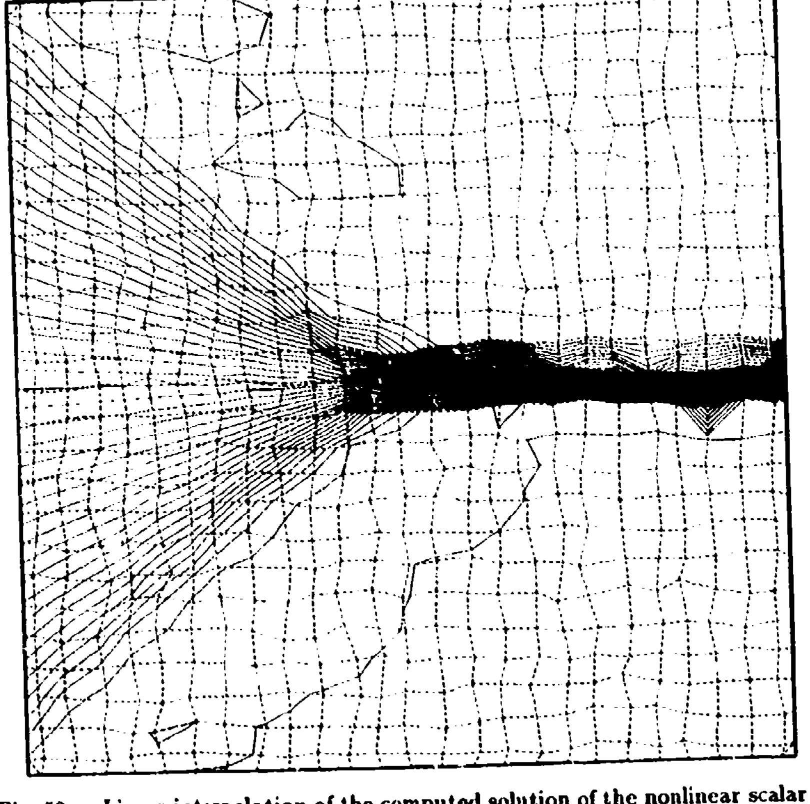


Fig. 52: Linear interpolation of the computed solution of the nonlinear scalar convection problem, (k=2, GP=2, GC-SSG).

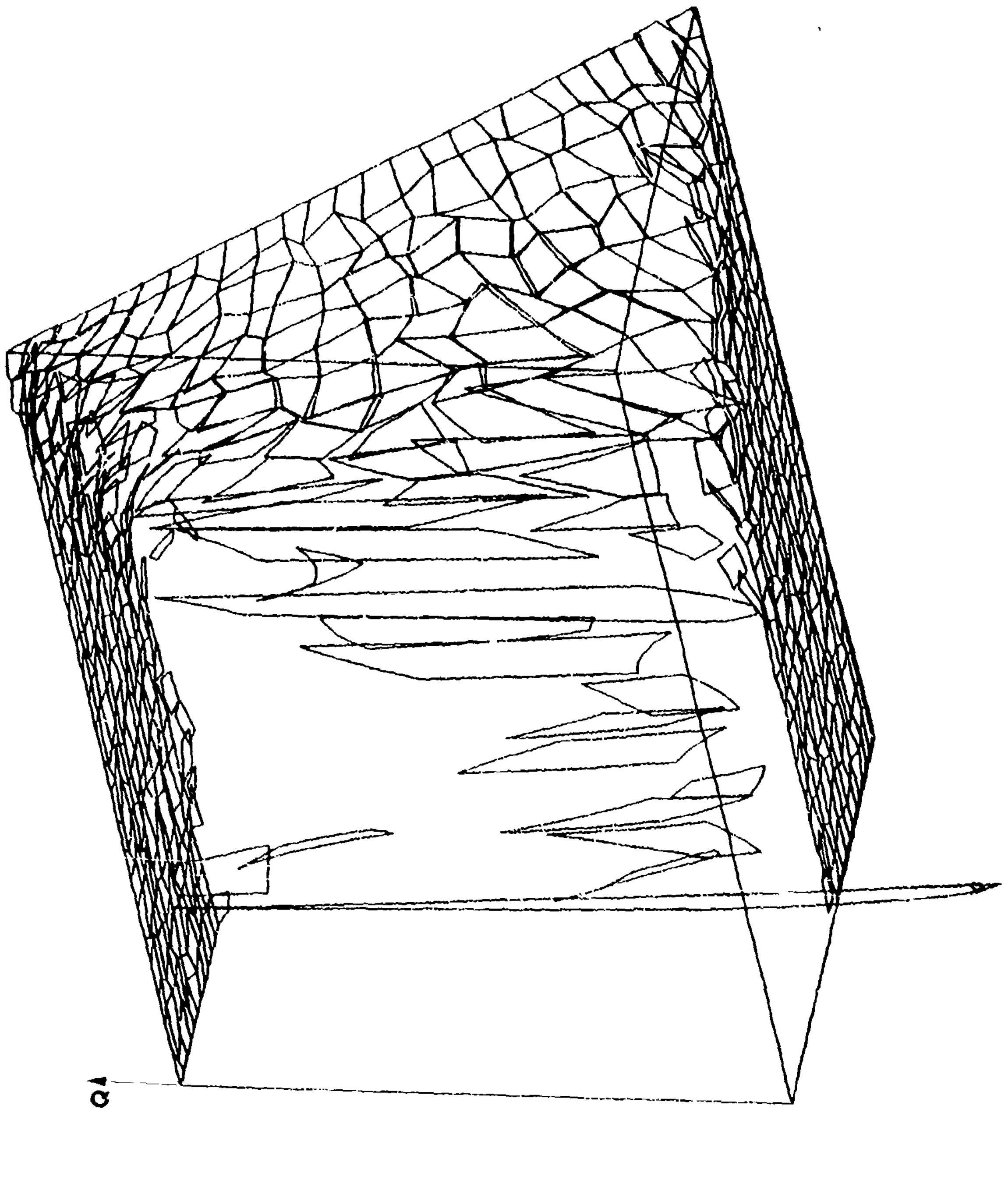


Fig. 53: Linear interpolation of the computed solution of the nonlinear scalar convection problem, (k=2, GP=2, GC-SSG).

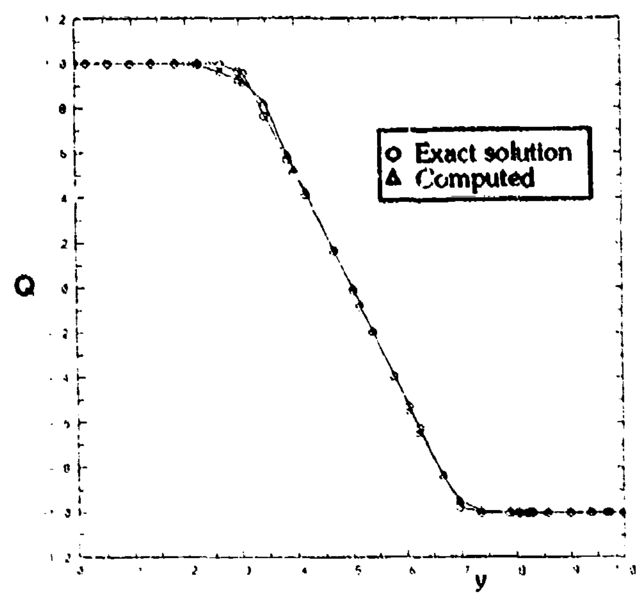


Fig. 54: Solution cut at x = 0.3 (k=2, GP=2, GC-SSG).

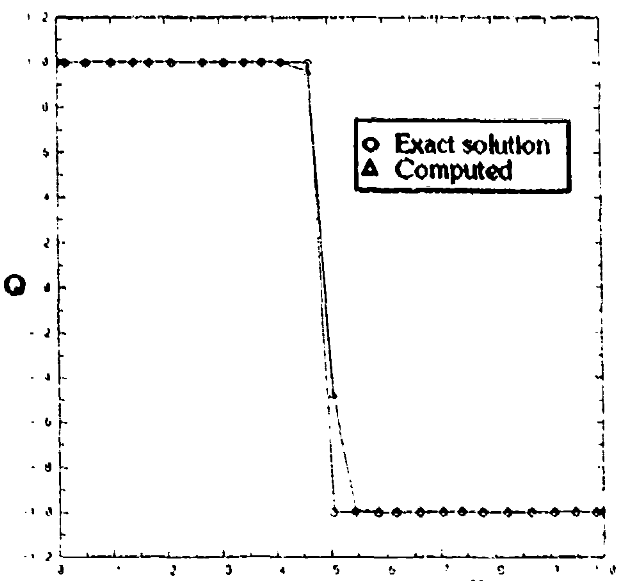


Fig. 55: Solution cut at x = 0.95 (k=2, GP=2, GC-SSG).

line with $\bar{q}$ = 0.3. The mesh used on this geometry is depicted in fig. 58 and contains about 600 triangular cells generated by a code written by J.D. Müller at the University of Michigan and which uses a combination of Delaunay and frontal methods, see ref. [43]. The flow turns from left to right.

8.4.2 Scheme

A fixed central support was used in the calculation because of its enhanced accuracy in the reconstruction phase (see [16]) and because of the absence of discontinuities in the flow. The solution in the form of conservative variables was reconstructed using a vector reconstruction; i.e. all variables use one and the same support. The values of the second order Zero-Mean reconstruction on both sides of each cell edge were passed into Van Leer's flux splitter to evaluate the (non-linear) flux through the current edge using one point (GP=1) Gauss quadrature. The CFL-number is 0.2 combined with a 4 step Runge-Kutta time stepping with the following coefficients : $(\frac{1}{4}, \frac{1}{3}, \frac{1}{2}, 1)$.

8.4.3 Boundary Conditions

The components of the unit vector tangent to the exactly known flow direction are imposed at the inlet. Note that these components vary along the inlet. Besides this, also the total pressure (1.2 bar) and the total temperature (298 K) are needed in order to treat a subsonic inlet boundary condition. Remark that the flow is inviscid and subsonic as a consequence of which there are no losses. Therefore, the total pressure and temperature can be assumed constant along the inlet. Zero normal velocity is required at the solid walls. Although this condition is sufficient to perform a correct boundary condition treatment, also a constant total pressure (1.2 bar) was imposed on the walls. This type of treatment was proposed by Denton in [44] and leads to a more accurate solution at the solid boundaries when a fully reversible or isentropic flow is present. As the speed is constant at the outlet as well as the total pressure, a constant static pressure (1.125 bar) was applied at the outlet. This pressure corresponds with the imposed total pressure at the inlet and with the parameter $\bar{q}$ = 0.3.

8.4.4 Solution

The exact solution presented on the mesh of fig. 58 using a 3^{rd} order reconstruction (k=3) is given in figures 59 and 60

The velocity lines obtained from the solution are shown in the plot of fig. 61.

It is clear that the tachlines are no perfect circles in the entire domain. However, the plot of constant density lines given in fig. 62 is comparable to the solution presented by Barth in fig. 63 obtained from ref. [45].

A further comparison between the density isolines of the numerical solution and the exact solution (fig. 62 and 59) shows a rather good agreement. The relative L_2-norm of the error as computed with eq (96) between the numerical solution and the exact one over the whole domain is of order 1% for the density. Fig. 64 presents the local relative L_2-norm of the error.

The largest errors are made at the inner wall of the duct where the higher order derivatives of the exact solution are higher and hence cannot be matched exactly by a quadratic reconstruction. Another reason is that the exact boundary is approximated by a series of straight boundary edges of more or less constant length. This approximation is of course less accurate where the curvature of the boundary is higher.

To show the improvement achieved with quadratic reconstruction with respect to a HOR-scheme using linear reconstruction, fig. 65 shows the tachlines obtained when using linear reconstruction and 1 Gauss point.

The underlying mesh is much more visible in the case of linear reconstruction. The relative L_2-norm of the density error over the entire domain is about 1.5% compared to the 1.0% obtained with quadratic reconstruction.

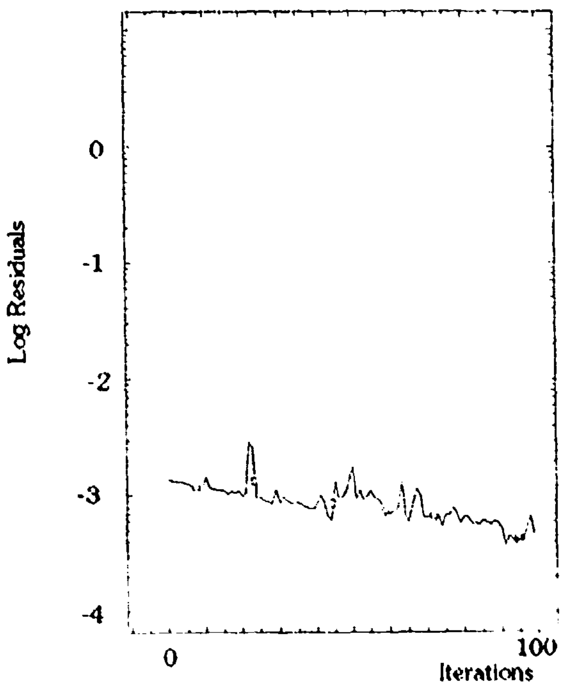


Fig. 56: Convergence history for nonlinear scalar convection problem, (k=2, GP=2, GC-SSG).

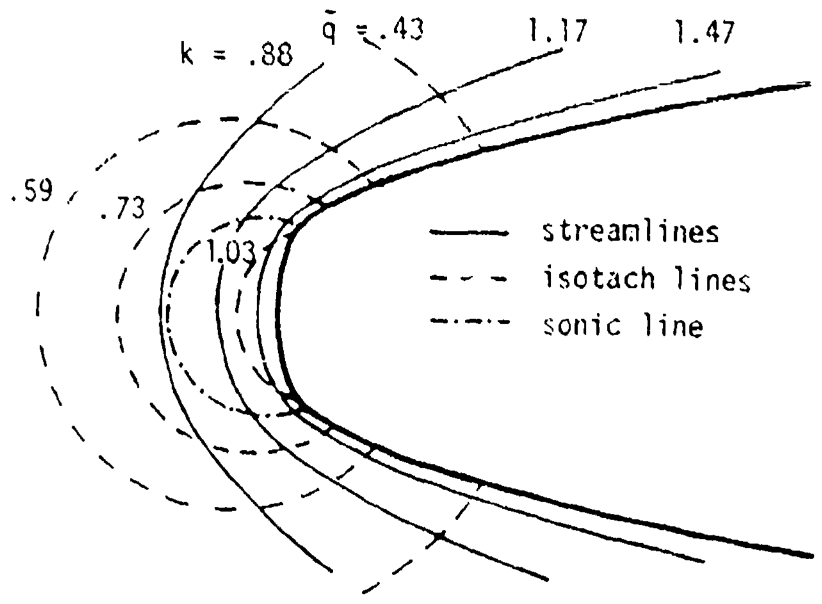


Fig. 57: Exact solution of Ringleb Flow.

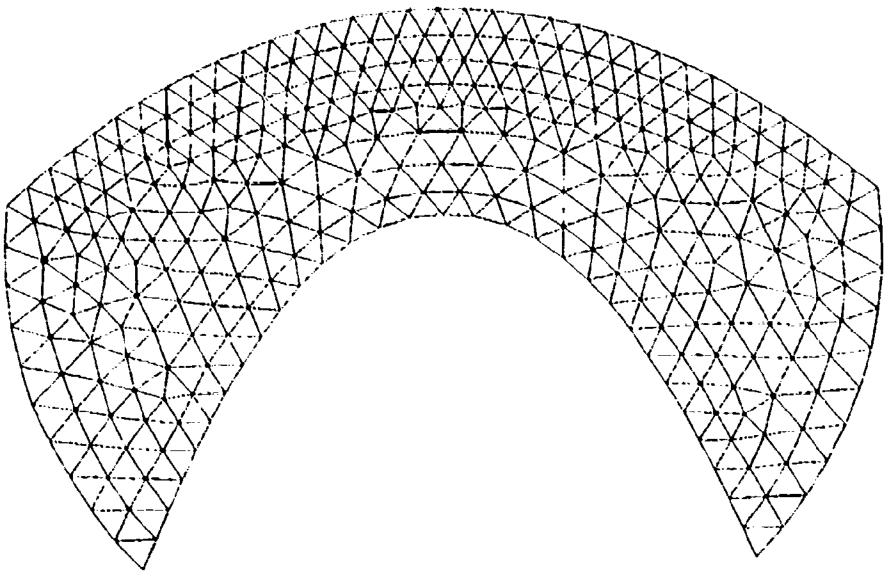


Fig. 58: Triangular Mesh for Ringleb Flow.

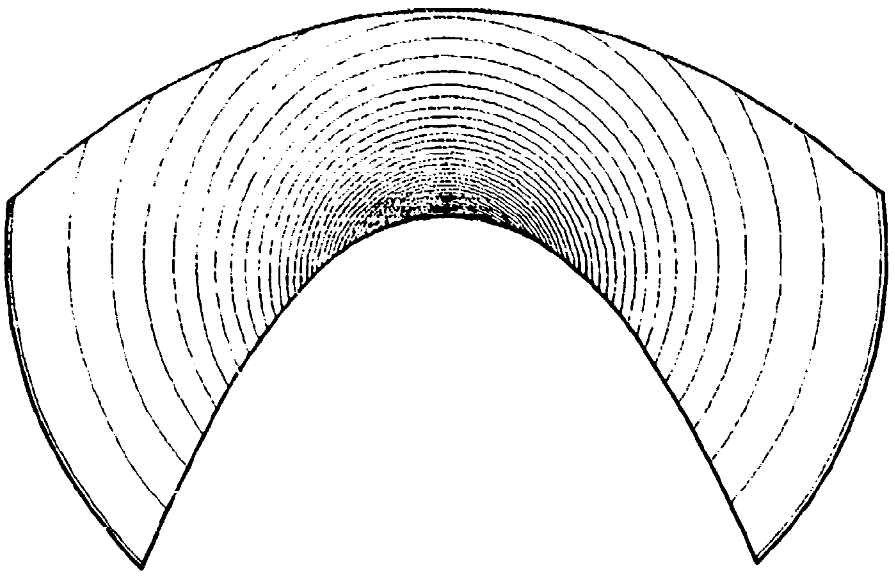


Fig. 59: Exact density contours of Ringleb flow (k=3).

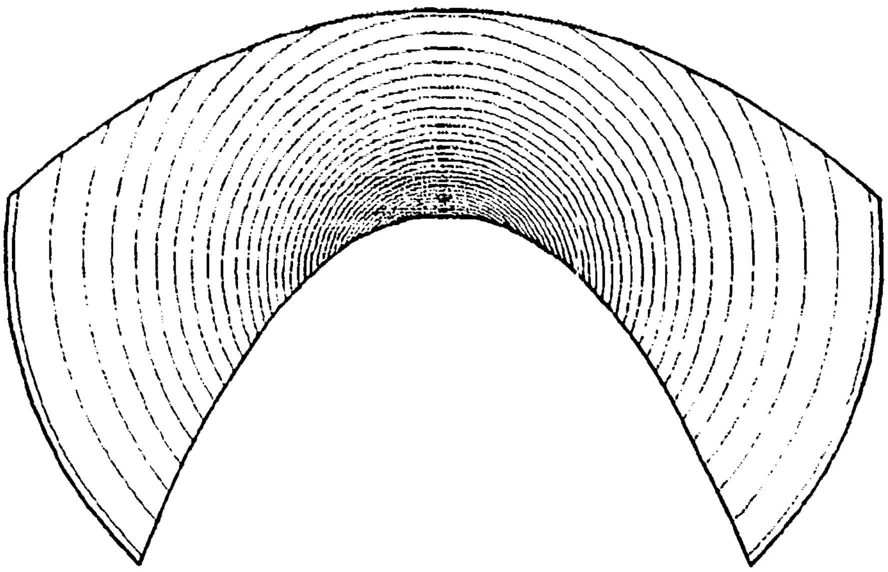


Fig. 60: Exact tachlines for Ringleb flow (k=3).

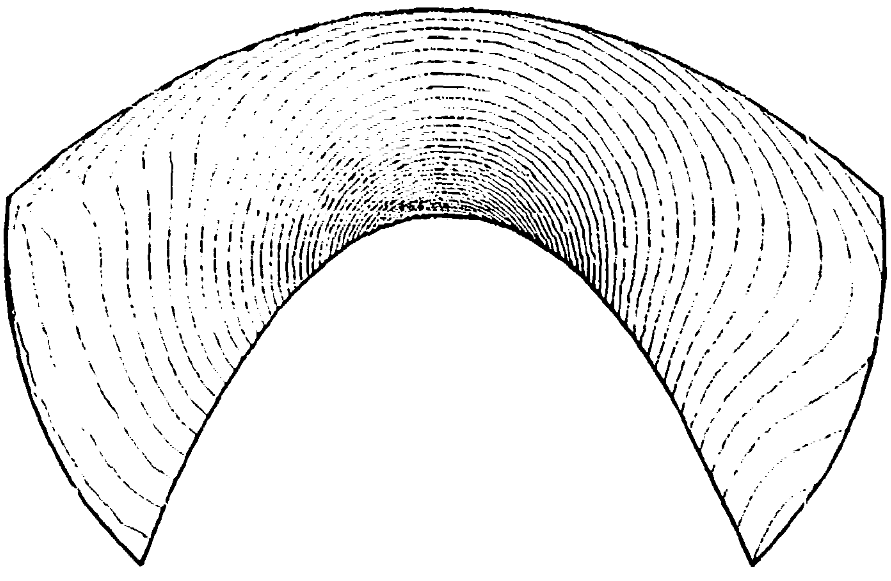


Fig. 61: Tachlines for Ringleb flow (k=2,GP=1).

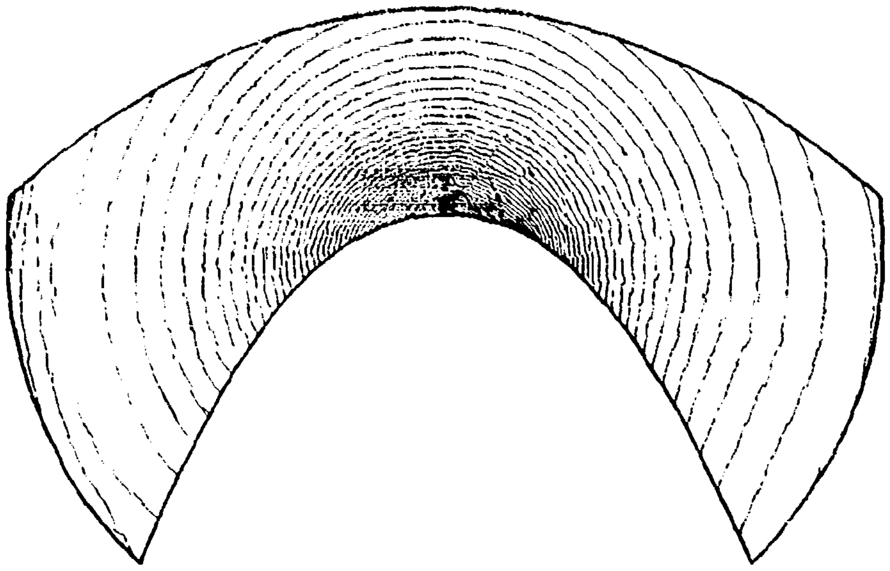


Fig. 62: Density lines for Ringleb flow (k=2,GP=1).

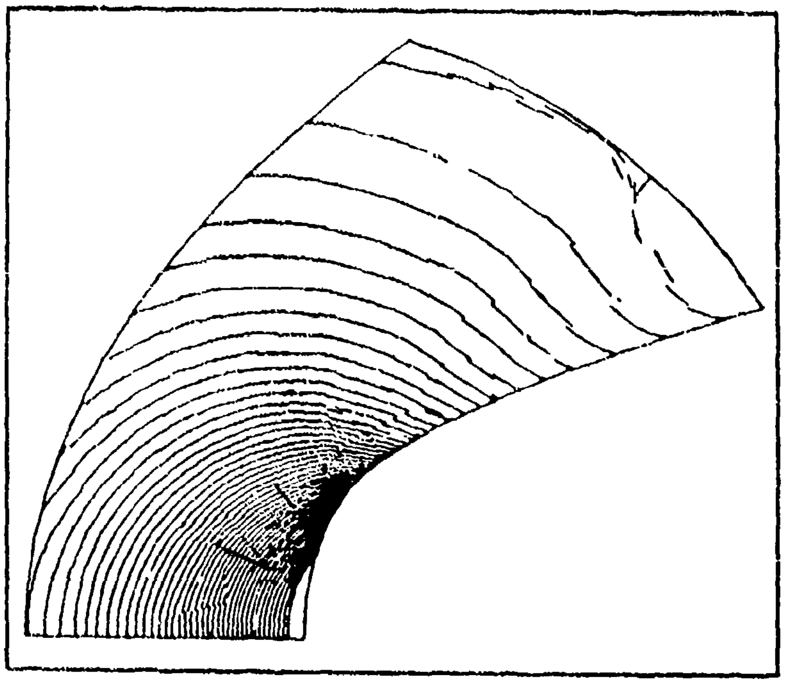


Fig. 63: Tachlines for Ringleb low by Barth (k=2,GP=1).

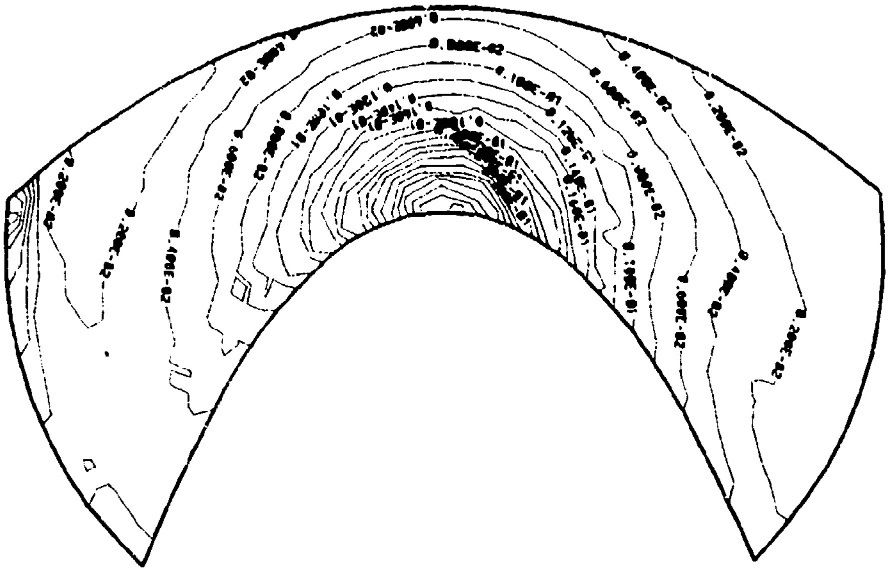


Fig. 64: Local relative L_2-norm of the density error (k=2,GP=1).

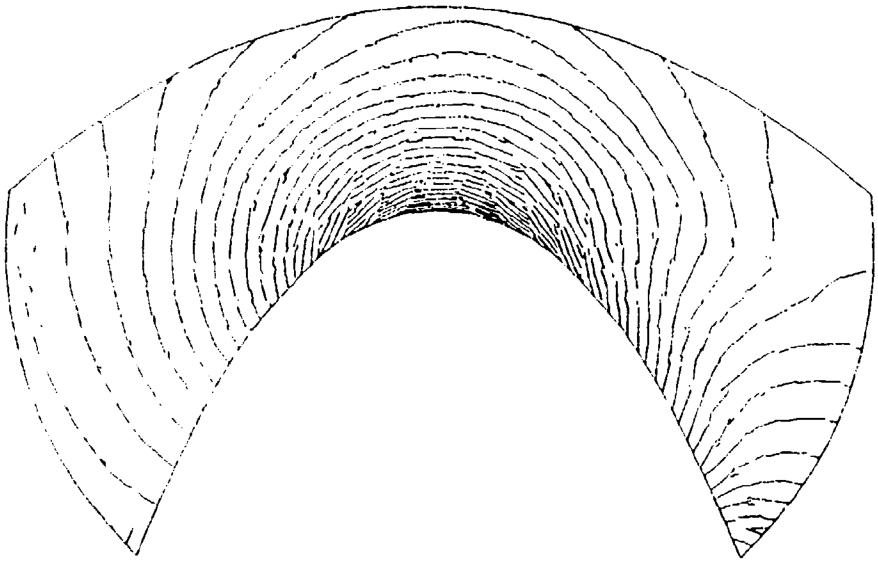


Fig. 65: Velocity lines for Ringleb flow (k=1,GP=1).

References

[1] VANKEIRSBILCK P., STRUIJS R., and DECONINCK H.,. "Solution of the Euler equations using unstructured polygonal meshes". In R. Gruber, J. Periaux, and R.P. Shaw, editors, *Proc. of the Fifth Int. Symp. on Num. Meth. in Engineering*, Lausanne, Switzerland, 11-15 Sept. 1989. Comp. Mech. Publications, Springer Verlag.

[2] VANKEIRSBILCK P. and DECONINCK H.,. "Computation of inviscid supersonic flows using adaptive meshes". In DESIDERI and PERIAUX J., editors, *Proc. of the Int. Workshop on Hypersonic Flows for Reentry Problems*, Antibes, France, 22-25 Jan. 1990. Springer Verlag. to be published.

[3] STRUIJS R., VANKEIRSBILCK P., and DECONINCK H.,. "An adaptive grid polygonal Finite Volume method for the compressible flow equations". In AIAA, editor, *AIAA 9th Comp. Fluid Dyn. Conf.*, pages 303-311, 1989. Paper AIAA-89-1959-CP.

[4] YEE H.C.,. "A class of high-resolution explicit and implicit shock-capturing methods". In *Proc. of the VKI Lecture Series in Comp. Meth. in Fluid Dynamics*, 6-10 March 1989. VKI LS 1989-04.

[5] YEE H.C., WARMING R.F., and HARTEN A.,. "Implicit Total Variation Diminishing (TVD) schemes for steady state solutions". *J. of Comp. Phys.*, 57-3:327-360, Feb. 1985.

[6] SPEKREIJSE S.P.,. *"Multigrid solutions of the steady Euler equations"*. PhD thesis, TU Delft, 1987.

[7] BARTH T.J. and FREDERICKSON P.O.,. "Higher order solution of the Euler equations on unstructured grids using quadratic reconstruction". *AIAA*, (90-0013), 8-11 Jan. 1990.

[8] BARTH T.J.,. "Aspects of unstructured grids and FV-solvers for the Euler and Navier-Stokes equations". In *Proc. of the AGARD-FDP-VKI Special Course*, 18-22 May 1992. AGARD-R-787.

[9] ESSERS J.A. and RENARD R.,. "An implicit flux-vector splitting Finite-Element technique for an improved solution of the compressible Euler equations on distorted grids". In *Proc. of the 11th Int. Conf. on Num. Meth. in Fluid Dyn.*, Williamsburg, Virginia, June 27 - July 1 1988.

[10] STRUIJS R. and DECONINCK H.,. "Fluctuation splitting schemes for the 2D Euler equations". In *Proc. of the VKI Lecture Series on Comp Fluid Dyn.*, Feb. 18 22 1991. VKI-LS 1991-01.

[11] HEMKER P.W. and KOREN B.,. "Efficient multidimensional upwinding for the steady Euler equations". Report NM-R9107, CWI, 1991.

[12] HIRSCH Ch.,. "A general analysis of two-dimensional convection schemes". In *Computational Fluid Dynamics*. von Karman Institute, 1991. VKI-LS 1991-01.

[13] HARTEN A., ENGQUIST B., and CHAKRAVARTHY S.R.,. "Uniformly high order accurate Essentially Non-Oscillatory schemes III". Technical Report 86-22, ICASE, April 1986.

[14] SHU C. and OSHER S.,. "Efficient implementation of Essentially Non-Oscillatory shock-capturing schemes, II". *J. of Comp. Phys.*, 83:32-78, 1989.

[15] DURLOFSKY L.J., OSHER S., and ENGQUIST B.,. "Triangle based TVD schemes for hyperbolic conservation laws". *J. of Comp. Phys.*, 1:64-73, Jan. 1992.

[16] HARTEN A. and CHAKRAVARTHY S.R.,. "Multi-dimensional ENO schemes for general geometries". Technical Report 91-76, ICASE, Sept. 1991.

[17] ANDERSON W.K., THOMAS J.L., and Van LEER B.,. "Comparison of Finite Volume flux vector splittings for the Euler equations". *AIAA-Journal*, 24(9):1453-1460, Sept. 1986.

[18] Van LEER B.,. "Flux-vector splitting for the Euler equations". In *Proc. 8th Int. Conf. on Numer. Meth. in Fluid Dynamics*, volume 170, pages pp 507-512. Springer Verlag, 1982.

[19] ROE P.L.,. "The approximate Riemann solvers, parameter vectors and difference schemes". *J. of Comp. Phys.*, 43(2):357-372, Oct. 1981.

[20] FROMM J.E.,. "Practical investigation of convective difference approximations of reduced dispersion". *The Physics of Fluids Supplement II*, 1969.

[21] DECONINCK H.,. "A survey of upwind principles for the multidimensional Euler equations". In *Proc. of VKI Lecture Series on Comp. Meth. in Fluid Dynamics*, 1987. VKI-LS 1987-04.

[22] DECONINCK H. and STRUIJS R.,. "Consistent boundary conditions for cell centered upwind Finite Volume Euler solvers". *Num. Meth. for Fluid Dynamics III, Clarendon Press, Oxford*, April 1988-12/CFD.

[23] DEGREZ G.,. "VKI TN: A unified wall boundary treatment for cell-centered Finite Volume discretizations of the Euler and Navier-Stokes equations". in preparation, 1991.

[24] POWELL K.G., BEER M.A., and LAW G.W.,. "An adaptive embedded mesh procedure for leading edge vortex flows". *AIAA*, (89-0080), 1989.

[25] DENTON J.,. "Analytical supersonic staggered wedge cascade, a test case for inviscid 2D flow calculations". In *Proc. of the VKI Lecture Series on Num. Meth. for Flows in Turbomachinery*, 22- 26 May 1989. VKI-LS 1989-06.

[26] HOLMES D.G.,. "2D inviscid test case results". In *Proc. of the VKI Lecture Series on Num. Meth. for Flows in Turbomachinery*, 22- 26 May 1989. VKI-LS 1989-06.

[27] DAHLQUIST G. and BJÖRCK Å.,. *"Numerical Methods"*. Prentice-Hall, 1974.

[28] DE MEYER H., VANDEN BERGHE G., and VANTHOURNOUT J.,. "On a new type of mixed interpolation". *J. of Comp. and Appl. Math.*, (30):pp 55-69, 1990.

[29] VANTHOURNOUT J., VANDEN BERGHE G., and DE MEYER H.,. "Families of backward differentiation methods based on a new type of mixed interpolation". *Computers Math. Appl.*, 20(11):19-30, 1990.

[30] HIRSCH Ch.,. *"Numerical Computation of Internal and External Flows"*, volume 1. Wiley John and Sons, 1988.

[31] SHU C. and OSHER S.,. "Efficient implementation of Essentially Non-Oscillatory shock-capturing schemes". *J. of Comp. Phys.*, 77:439–471, 1988.

[32] LEVEQUE R.J.,. "The accuracy of conservative methods for the advection equation on nonuniform grids". priv. comm., October 5 1987.

[33] JOHNSON C.,. *"Numerical solution of partial differential equations by the Finite Element method"*. Studentlitteratur, 1987.

[34] MIZUKAMI A. and HUGHES T.J.R.,. "A Petrov-Galerkin Finite Element method for convection dominated flows: an accurate upwinding technique for satisfying the maximumprinciple". *Comp. Meth. Appl. Mech. and Eng.*, 50(2):181–194, 1985.

[35] HARTEN A.,. "High resolution schemes for hyperbolic conservation laws". *J. of Comp. Phys.*, 49:357–393, 1983.

[36] HARTEN A.,. "Recent developments in shock-capturing schemes". Technical Report 91-08, ICASE, Jan. 1991.

[37] SHU C.-W.,. "Numerical experiments on the accuracy of ENO and modified ENO schemes". Technical Report 90-55, ICASE, Aug. 1990.

[38] HARTEN A.,. "ENO schemes with subcell resolution". NASA CR 178 362 87-56, ICASE, Aug. 1987.

[39] ABGRALL R.,. "Design of an Essentially Non-Oscillatory reconstruction procedure on Finite Element type meshes". Technical Report 91-84, ICASE, Dec. 1991.

[40] YANG H.,. "An artificial compression method for ENO schemes: the slope method". *J. of Comp. Phys.*, 89:125–160, 1990.

[41] RINGLEB F.,. "Exakte Lösungen der Differentialgleichungen einer adiabatischen Gasströmung". *ZAMM*, 20(4):185–198, August 1940.

[42] CHIOCCHIA G.,. "Exact solutions to transonic and supersonic flows". Technical Report AR-211, AGARD, 1985.

[43] MULLER J.-D., ROE P.L., and DECONINCK H.,. "A frontal approach for node generation in Delaunay algorithms". In *Proc. of the AGARD-FDP-VKI Special Course*, 18-22 May 1992. AGARD-R-787.

[44] DENTON J.D.,. "An improved time-marching method for turbomachinery flow calculation". *J. Eng. for Power*, 105(3):514-524, July 1983.

[45] BARTH T.J.,. "On unstructured grids and solvers". In *Proc. of VKI Lecture Series on Comp. Fluid Dynamics*, 5-9 March 1990. VKI-LS 1990-03.

A Proofs of Higher Accuracy

Proof of Theorem 5.1 (A):

To proof order h^k space accuracy one has to investigate the local truncation error. The truncation error is the part in the equivalent differential equation that is not present in the analytical modelling equation to be solved. To obtain the equivalent differential equation, the unknowns in the stencil cells of the employed scheme are developed in a Taylor series around some point $\vec{r}_P$ assigned to the cell where the truncation error is to be evaluated, see [30] and [12]. The differential equation is the equation which is satisfied by the exact solution q in a discrete set of points $\vec{r}_P$ around which the Taylor expansions have been made. As yet, the choice of the point $\vec{r}_P$ remains arbitrary.

In this proof, the procedure deviates a little from the classic one given in [30] and [12] in that it first uses the theorems 4.1 and 4.3 before applying Taylor series expansions. Thus, using eqs. (92) or (93) and assuming that the exact solution q of the equivalent differential equation varies smoothly, the space discretization of scheme (101) can then be rewritten as:

$$\begin{aligned}\left[D(\vec{a}.\vec{\nabla}q)\right](q) = &\\ &\frac{1}{\Omega_P}.\sum_R\sum_O\left\{a^+_{PR}.\left[q(\vec{r}_O)+\mathcal{O}(h^{k+1})\right]\right.\\ + &\left.a^-_{PR}.\left[q(\vec{r}_O)+\mathcal{O}(h^{k+1})\right]\right\}.\Delta s_O\end{aligned}$$

which, in its turn, is equal to

$$\begin{aligned}\left[D(\vec{a}.\vec{\nabla}q)\right](q) = &\\ &\frac{1}{\Omega_P}.\sum_R\sum_O(\vec{a}.\vec{n}_{PR}).q(\vec{r}_O).\Delta s_O + \mathcal{O}(h^k)\mathcal{D}_q^{k+1}\end{aligned}$$

The value of $q(\vec{r}_O)$ can now be replaced by a Taylor series expansion around the reference point $\vec{r}_P$ while truncating it after the the terms of order k:

$$\begin{aligned}\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}.\sum_R\sum_O(\vec{a}.\vec{n}_{PR}).&\\ &\left[q_P+\sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{q}_l^P\right].\Delta s_O\\ + \ \mathcal{O}(h^k)\mathcal{D}_q^{k+1}&\end{aligned}$$

where the definition of $\mathbf{C}_l^P$ is given in eq. (90) while the one of $\partial\mathbf{q}_l^P$ is similar to $\partial\mathbf{Q}_l^P$ as found in eq. (69) but now applied to the exact solution q of the numerical scheme.

As $(\vec{a}.\vec{n}_{PR})$ is constant along each straight edge, the previous expression for the space discretization can be rewritten as a continuous contour integral if the number of Gauss points O on the edges is larger than or equal to $\frac{k+1}{2}$:

$$\begin{aligned}\left[D(\vec{a}.\vec{\nabla}q)\right](q) = &\\ &\frac{1}{\Omega_P}.\oint_{\partial\Omega_P}(\vec{a}.\vec{n}).\left[q_P+\sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}).\partial\mathbf{q}_l^P\right].ds\\ &+\mathcal{O}(h^k)\mathcal{D}_q^{k+1}\end{aligned}$$

or

$$\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}.\oint_{\partial\Omega_P}(\vec{a}.\vec{n}).q.ds + \mathcal{O}(h^k)\mathcal{D}_q^{k+1}$$

Using Green's theorem, one finds finally that the space discretization satisfies the following equation:

$$\left[D(\vec{a}.\vec{\nabla}q)\right](q) = <\vec{a}.\vec{\nabla}q>^P + \mathcal{O}(h^k)\mathcal{D}_q^{k+1} \qquad (165)$$

which completes the first proof for order h^k space accuracy. Note also that one is discretizing mean values over a cell even when the GC-HOR-Algorithm is used. Point values are discretized only if the order of the reconstruction $k \leq 1$.

□

Proof of Theorem 5.1 (B):

The second proof follows rigorously the classic truncation error analysis as described in [30] and [12]. This means that all solution values in the support cells have to be expanded as well in the vector $\overline{\mathbf{Q}}$ or $\mathbf{Q}$ as defined in eq. (39) resp. (80).

The proof will only be given here for the scheme using ZM-HOR extrapolation since the reasoning for the GC-HOR scheme is completely similar and leads to the same conclusions. The proof will rely heavily on the properties of the reconstruction weights as expressed in eqs. (64), (63), (81) and (83).

To examine the local truncation error, the expressions (104) and (105) have to be developed in a Taylor series and plugged into eq. (101). The Taylor expansions of the mean of the solution in cell P and a cell R next to cell P are given by the expressions (71) resp. (72). The reference point $\vec{r}_P$ in these expressions is the same as the one used to compute the inertial moments when reconstructing the solution, eq. (33). The properties expressed by eq. (64) can then immediately be exploited when applying eq. (71) to the scheme given by eq. (101) with q the exact solution of the equivalent differential equation:

$$\begin{aligned}
\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}\sum_R\sum_O \Delta s_O \\
\Bigg\{a^+_{PR}\cdot\Bigg[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{I}_l^{P,P}.\partial\mathbf{q}_l^P \\
+ \mathbf{W}^P.\left(\mathbf{q}_P + \sum_{l=1}^{k}\frac{1}{l!}\mathcal{I}_l^{\bullet,P}.\partial\mathbf{q}_l^P\right).\tilde{\mathbf{F}}^P\Bigg] \\
+a^-_{PR}\cdot\Bigg[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{I}_l^{R,P}.\partial\mathbf{q}_l^P \\
+ \mathbf{W}^R.\left(\mathbf{q}_P + \sum_{l=1}^{k}\frac{1}{l!}\mathcal{I}_l^{\prime,P}.\partial\mathbf{q}_l^P\right).\tilde{\mathbf{F}}^R\Bigg]\Bigg\} \\
+ \mathcal{O}(h^k)\mathcal{D}_q^{k+1} \qquad (166)
\end{aligned}$$

where

$$\mathbf{q}_P = [q_P \cdots q_P]^T \ .$$

$$\mathcal{I}_l^{\bullet,P} = \left[\mathbf{I}_l^{R_1,P}\ldots\mathbf{I}_l^{R_i,P}\ldots\mathbf{I}_l^{R_n,P}\right]^T$$

and

$$\mathcal{I}_l^{\prime,P} = \left[\mathbf{I}_l^{R'_1,P}\ldots\mathbf{I}_l^{R'_i,P}\ldots\mathbf{I}_l^{R'_n,P}\right]^T$$

with R'_i a support cell of the direct neighbour R of cell [illegible] Using now the properties (64) of the reconstruction [illegible]hts, eq. (166) becomes:

$$\begin{aligned}
\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}\sum_R\sum_O \Delta s_O. \\
\Bigg\{a^+_{PR}\cdot\left[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{q}_l^P\right] \\
+a^-_{PR}\cdot\Bigg[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{I}_l^{R,P}.\partial\mathbf{q}_l^P \\
+ \sum_{l=1}^{k}\frac{1}{l!}\mathbf{W}^R.\mathcal{I}_l^{\prime,P}.\partial\mathbf{q}_l^P.\tilde{\mathbf{F}}^R\Bigg]\Bigg\} \\
+ \mathcal{O}(h^k)\mathcal{D}_q^{k+1} \qquad (167)
\end{aligned}$$

Keeping in mind eq. (64), it is desirable to rewrite $I_{l-m,m}^{R',P}$ in terms of $I_{s,t}^{R',R}$ with $0 \leq s \leq l$ and $0 \leq t \leq m$ and R' a cell belonging to the support of cell R. This can be done using the next transformation formula:

$$\begin{aligned}
I_{l-m,m}^{R',P} = \\
\sum_{s=0}^{l-m}\sum_{t=0}^{m}\binom{l-m}{s}\cdot\binom{m}{t}\cdot \\
\Delta x_{PR}^{(l-m)-s}.\Delta y_{PR}^{t}.I_{s,t}^{R',R} \qquad (168)
\end{aligned}$$

Using eq. (168) and taking into account the properties given by eq. (64), one finds after some manipulations that:

$$\mathbf{W}^R.\mathcal{I}_l^{\prime,P} = \mathbf{C}_l^P - \mathbf{I}_l^{R,P}$$

With these results the space discretization becomes:

$$\begin{aligned}
\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}\sum_R\sum_O \Delta s_O. \\
\Bigg\{a^+_{PR}\cdot\left[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{q}_l^P\right] \\
+ a^-_{PR}\cdot\left[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{q}_l^P\right]\Bigg\} \\
+\mathcal{O}(h^k)\mathcal{D}_q^{k+1} \\
= \frac{1}{\Omega_P}\sum_R\sum_O(\vec{a}.\vec{n}_{PR}). \\
\left[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}_O).\partial\mathbf{q}_l^P\right].\Delta s_O \\
+\mathcal{O}(h^k)\mathcal{D}_q^{k+1}
\end{aligned}$$

Again, as $(\vec{a}.\vec{n}_{PR})$ is constant along each straight edge, the previous expression for the space discretization can be rewritten as a continuous contour integral if the number of Gauss points O on the edges is larger than equal to $\frac{k+1}{2}$:

$$\begin{aligned}
\left[D(\vec{a}.\vec{\nabla}q)\right](q) = \frac{1}{\Omega_P}\cdot\oint_{\partial\Omega_P}(\vec{a}.\vec{n}). \\
\left[q_P + \sum_{l=1}^{k}\frac{1}{l!}\mathbf{C}_l^P(\vec{r}).\partial\mathbf{q}_l^P\right].ds \\
+ \mathcal{O}(h^k)\mathcal{D}_q^{k+1}
\end{aligned}$$

From now on, the reasoning proceeds in exactly the same way as in the first proof leading to eq. (165).

□

B First Order Accuracy

Theorem B.1
First order accuracy can be obtained on irregular meshes when only cell-wise constant solution reconstruction is used.

Proof of Theorem B.1:

Let us first write down the truncation error T_P^n in cell P at time level n for the linear scheme given by eq. (101) where now $Q_{PO}^{(k)} = q_P$ and $Q_{RO}^{(k)} = q_R$ with q the exact solution of the equivalent differential equation:

$$\begin{aligned} T_P^n = \\ & q_t|_P^n + \frac{\Delta t}{2}\, q_{tt}|_P^n + \mathcal{O}(\Delta t^2) \\ + & \frac{1}{\Omega_P}\sum_R\sum_O \{a_{PR}^+.q_P^n \\ & + a_{PR}^-\,[q_P^n + \Delta x_{PR}.\, q_x|_P^n + \Delta y_{PR}.\, q_y|_P^n \\ & + \frac{1}{2}\Delta x_{PR}^2.\, q_{xx}|_P^n + \frac{1}{2}\Delta y_{PR}^2.\, q_{yy}|_P^n \\ & + \Delta x_{PR}.\Delta y_{PR}.\, q_{xy}|_P^n]\}\,.\Delta s_O \\ + & \mathcal{O}(h^2) \end{aligned}$$

From eq. (97) it follows that:

$$\begin{aligned} q_t|_P^n &= -a.\, q_x|_P^n - b.\, q_y|_P^n \\ q_{xt}|_P^n &= -a.\, q_{xx}|_P^n - b.\, q_{xy}|_P^n \\ q_{yt}|_P^n &= -a.\, q_{xy}|_P^n - b.\, q_{yy}|_P^n \\ q_{tt}|_P^n &= a^2.\, q_{xx}|_P^n + 2ab.\, q_{xy}|_P^n + b^2.\, q_{yy}|_P^n \end{aligned}$$

and furthermore:

$$\begin{aligned} & \sum_R\sum_O (a_{PR}^+.q_P^n + a_{PR}^-.q_P^n)\,.\Delta s_O \\ &= q_P^n.\sum_R\sum_O \vec{a}.\vec{n}_{PR}.\Delta s_O \\ &= q_P^n.\oint_{\partial\Omega_P} \vec{a}.\vec{n}.ds \\ &= 0 \end{aligned}$$

so that T_P^n becomes:

$$\begin{aligned} T_P^n = \\ & \left[\frac{1}{\Omega_P}\sum_R\sum_O a_{PR}^-.\Delta x_{PR}.\Delta s_O - a\right].\, q_x|_P^n \\ + & \left[\frac{1}{\Omega_P}\sum_R\sum_O a_{PR}^-.\Delta y_{PR}.\Delta s_O - b\right].\, q_y|_P^n \\ + & \left[\frac{1}{2\Omega_P}\sum_R\sum_O a_{PR}^-.\Delta x_{PR}^2.\Delta s_O + a^2\frac{\Delta t}{2}\right].\, q_{xx}|_P^n \\ + & \left[\frac{1}{2\Omega_P}\sum_R\sum_O a_{PR}^-.\Delta y_{PR}^2.\Delta s_O + b^2\frac{\Delta t}{2}\right].\, q_{yy}|_P^n \\ + & \left[\frac{1}{\Omega_P}\sum_R\sum_O a_{PR}^-.\Delta x_{PR}.\Delta y_{PR}.\Delta s_O \right. \\ & \left. + ab.\Delta t\right].\, q_{xy}|_P^n \\ + & \mathcal{O}(h^2) + \mathcal{O}(\Delta t) \end{aligned}$$

As the scheme is only stable if the CFL-number is smaller than one, it follows that $\Delta t = \mathcal{O}(h)$. Hence, all coefficients of the higher order derivatives are $\mathcal{O}(h)$ and one gets:

$$T_P^n = \left[\frac{1}{\Omega_P}\sum_R\sum_O a_{PR}^-\,\vec{r}_{PR}.\Delta s_O - \vec{a}\right].\,\vec{\nabla}q\Big|_P^n + \mathcal{O}(h)$$

Knowing that

$$\frac{1}{\Omega_P}\sum_R\sum_O (\vec{a}.\vec{n}_{PR}).\vec{r}_{PO}.\Delta s_O = \vec{a}$$

if the number of Gauss points is at least equal to 1 and since

$$\vec{r}_{PR} = \vec{r}_{PO} - \vec{r}_{RO}$$

T_P^n can be rewritten as:

$$\begin{aligned} T_P^n = \\ - & \frac{1}{\Omega_P}\left[\sum_R\sum_O (a_{PR}^-.\vec{r}_{RO} + a_{PR}^+.\vec{r}_{PO})\,.\Delta s_O\right].\,\vec{\nabla}q\Big|_P^n \\ + & \mathcal{O}(h) \end{aligned}$$

Taking into account that

$$\vec{\nabla}q.\Big|_R^n = \vec{\nabla}q\Big|_P^n + \mathcal{O}(h)$$

one finds

$$\begin{aligned} T_P^n = \\ - & \frac{1}{\Omega_P}\sum_R\sum_O \left[a_{PR}^-.\vec{r}_{RO}\ \vec{\nabla}q\Big|_R^n \right. \\ & \left. + a_{PR}^+.\vec{r}_{PO}.\,\vec{\nabla}q\Big|_P^n\right].\Delta s_O \\ + & \mathcal{O}(h) \end{aligned}$$

Defining

$$\begin{aligned} \hat{T}_P^n &= \vec{r}_{PO}.\,\vec{\nabla}q\Big|_P^n \\ \hat{T}_R^n &= \vec{r}_{RO}.\,\vec{\nabla}q\Big|_R^n \\ \hat{T}_P^n &= \mathcal{O}(h) \end{aligned}$$

T_P^n can be regarded as the space discretization of $\hat{T}$ instead of q plus a term of order h:

$$\begin{aligned} T_P^n = \\ & -\frac{1}{\Omega_P}\sum_R\sum_O \left[a_{PR}^+.\hat{T}_P^n + a_{PR}^-.\hat{T}_R^n\right].s_O \\ & + \mathcal{O}(h) \\ = & \ D\left(\hat{T}_P^n\right) + \hat{T}_P^n \end{aligned} \tag{169}$$

The global error of the scheme of eq. (101) is now defined as:

$$E_P^n = \tilde{q}(\vec{r}_P, t_n) - q_P^n$$

with $\tilde{q}$ the exact solution of the analytical problem (97) and, due to the linearity of the scheme, the global error satisfies:

$$
\begin{aligned}
T_P^n = & \\
& \frac{1}{\Delta t}\left(E_P^{n+1} - E_P^n\right) \\
+ & \frac{1}{\Omega_P}\sum_R \sum_O \left(a_{PR}^{+}.E_P^n + a_{PR}^{-}.E_R^n\right).\Delta s_O
\end{aligned}
\qquad (170)
$$

or

$$
\frac{1}{\Delta t}\left(E_P^{n+1} - E_P^n\right) + D\left(E_P^n\right) = T_P^n \qquad (171)
$$

The global error can be decomposed as

$$
E_P^n = \hat{E}_P^n + \check{E}_P^n \qquad (172)
$$

where

$$
\begin{aligned}
\hat{E}_P^n &= \hat{T}_P^n \doteq \vec{r}_{PO}.\vec{\nabla}q\Big|_P^n = \mathcal{O}(h) \\
\check{E}_P^n &= E_P^n - \hat{E}_P^n
\end{aligned}
$$

Equation (171) becomes then:

$$
\begin{aligned}
& \frac{1}{\Delta t}\left(\check{E}_P^{n+1} - \check{E}_P^n\right) \\
&= \hat{T}_P^n - \frac{1}{\Delta t}\left(\hat{E}_P^{n+1} - \hat{E}_P^n\right) \\
&= \hat{T}_P^n - \frac{1}{\Delta t}\left(\hat{T}_P^{n+1} - \hat{T}_P^n\right) \\
&= \hat{T}_P^n - \frac{\vec{r}_{PO}}{2}.\left(\vec{\nabla}q_t\Big|_P^n + \mathcal{O}(\Delta t)\right) \\
&= \hat{T}_P^n + \mathcal{O}(h) = \mathcal{O}(h)
\end{aligned}
$$

Since the scheme is stable, one finds that the error $\check{E}_P^n$ remains $\mathcal{O}(h)$ over a fixed time interval. It follows then that the global error E_P^n is $\mathcal{O}(h)$. The scheme given by eq. (101) is therefore first order accurate in space and time.

□

FINITE ELEMENT METHODS IN CFD: GRID GENERATION, ADAPTIVITY AND PARALLELIZATION

by

Rainald Löhner
CMEE, SEAS, The George Washington University
Washington, D.C. 20052
United States

FOREWORD AND INTRODUCTION

Numerical methods for the solution of field problems using unstructured grids have reached a high degree of maturity. In Computational Fluid Dynamics (CFD), their impact has come relatively late. CFD has been traditionally dominated by structured grid solvers. These simpler solvers were used on relatively simple domains that were still of engineering interest, e.g. an airfoil or a wing. Only as the computational power of hardware increased to current levels did the possibility of computing geometrically complex designs become a reality. It was at this point that methods based on unstructured grids started to have an impact on mainstream CFD.

The following set of notes are part of a larger set that I had originally planned for this short course. As the final program only called for me to describe grid generators, adaptive refinement schemes, visualization and parallelization issues, I have restricted them to these topics. The topic of visualization has been left out entirely. Given the description of optimal data structures for grid generation, the reader can easily devise optimal search algorithms for visualization.

I have preceded these topics with a short chapter that places CFD among related disciplines, tries to define its aims, and focuses on the end-product of CFD research: flow simulation codes. The newcomer should find this section particularly interesting.

The reference section covers all topics originally intended for this course. The reader is encouraged to consult all of these for a more complete picture of CFD.

1. CFD: GENERAL CONSIDERATIONS

Before going into a detailed description of algorithms used for Computational Fluid Dynamics (CFD), it seems proper to place this discipline among related disciplines. CFD is part of Computational Mechanics, which in turn is part of Simulation Techniques. The aim is to approximate physically relevant situations and phenomena using computers. In CFD, this is accomplished by solving numerically Partial Differential Equations (PDEs), or by following the interaction of a large numbers of particles. Due to its relevance to the aerospace industry, as well as to most manufacturing processes, CFD has been pursued actively ever since the first digital computers were developed. The Manhattan project was a major testbed and beneficiary of early CFD technology. Concepts like artificial dissipation date from this time.

CFD, by its very nature, encompasses a variety of disciplines, which may be enumerated in the following order of importance:

- Engineering: We live in a technology-driven world. Engineering provides the reason why we pursue CFD. Forget the romantic vision of researchers mimicking art for art's sake. This is engineering, and if a code can not guide an engineer to better products, it is simply useless.
- Physics: Physics explains the phenomena to be simulated for engineering purposes, and provides possible approximations and simplifications to ab initio physics. For example, the potential approximation, where applicable, represents CPU savings of several orders of magnitude as compared to full Reynolds-Averaged Navier-Stokes (RANS) simulations. It is the task of this discipline to outline the domains of validity of the different assumptions and approximations that are possible.
- Mathematics: Mathematics has three different types of input for CFD applications. These are:
  a) Classic Analysis, which discusses the nature, boundary conditions, Green kernels, underlying variational principles, adjoint operators, etc. of the PDEs;
  b) Numerical Analysis, which describes the stability, convergence rates, uniqueness of solutions, well-posedness of numerical schemes, etc.; and
  c) Discrete Mathematics, which enables the rapid execution of arithmetic operations (try solving a square-root by hand).
- Computer Science: Computer science has mushroomed into many subdisciplines. The most important ones for CFD are:
  a) Algorithms, which describe how to perform certain operations in an optimal way (e.g. search of items in a list or in space);
  b) Coding, so that the final code is portable,

easy to modify and/or expand, easy to understand, user-friendly, etc.;

c) Software, which not only encompasses compilers, debuggers and operating systems, but also advanced graphics libraries (e.g. try doing what you can do with GL in PHIGS); and

d) Hardware, which drives not only the realm of ever expanding applications that would have been unthinkable a decade ago, but also influences to a large extent the algorithms employed and the way codes are written.

- Visualization Techniques: The vast amounts of data produced by modern simulations need to be displayed in a sensible way. This not only refers to optimal algorithms to filter and traverse the data at hand, but also to ways of seeing this data (plane-cuts, iso-surfaces, X-rays, stereo-vision, etc.).

- User Community: The final product of any CFD effort is a code that is to be used for engineering applications. Successful codes tend to have a user-community. This introduces human factors which have to be accounted for: confidence and benchmarking, documentation and education, the individual motivation of the end-users, ego-factors, not-invented-here syndrome, etc.

1.1 The CFD Code

The end-product of any CFD effort is a code that is to be used for engineering applications. The quality of this tool will depend on the quality of ingredients listed above. Just as a chain is only as strong as its weakest member, so is a code only as good as the worst of its ingredients. Given the breadth and variety of disciplines required for a good code, it is not surprising that only a few codes make it to a production environment, although so many are written worldwide. Once a CFD code leaves the realms of research, it becomes a tool, i.e. a part of the service industry. CFD codes, like other simulation codes, have certain properties. Some of these are:

- EU: Ease of Use (Problem set-up, User interface, ..)
- DO: Documentation (Manuals, Help, ..)
- GF: Geometric Flexibility
- TT: Turnaround Time (Set-up to end-result)
- BM: Benchmarking
- AC: Accuracy
- SP: Speed

As any other product, CFD codes have a customer base. This customer base may be categorized by the number of times a certain application has to be performed. Three main types of end-users may be identified:

a) Those that require a few runs on new configurations every so often to guide them in their designs (the general purpose run of manufacturing industries and process control);

b) Those that require a large number of runs to optimize highly sophisticated products (e.g. airfoil or wing optimization); and

c) Those that require a few very detailed runs on extremely simple geometries in order to understand or discover new physics (the NASA/NRL/LLNL/LANL/etc. scenario, where each run takes at least 500 hours of CRAY-time).

According to this frequency of runs, the priorities change, as can be seen from the following table:

Gen.Purp./Anal.	Des./Optim.	New Phys.
$O(1)$	$O(1,000)$	$O(10)$
days	seconds	months
EU	SP	AC
DO	TT	BM
GF	GF	SP
TT	AC	TT
BM	BM	EU
AC	EU	GF
SP	DO	DO

The message is clear: before comparing codes, ask how often the code is to be used on a particular application, how qualified the personnel is, what the maximum allowed turnaround time is, the expected accuracy and the resources available. Only then can a proper choice of codes be made.

1.2 Porting Research Codes to an Industrial Context

Going from a research code to an industrial code requires a major change of focus. Industrial codes are characterized by:

- Extensive manuals and other documentation;
- 24-hour hotline answering service;
- Customer support team for special requests/applications;
- Incorporation of changes through releases and training.

In short, they require an organization to support them. Many researchers (particularly members of academia who always longed for their own company) seem to ignore this. The result is a proliferation of small companies that neither satisfy customer needs, nor attain a high scientific level in their codes. Thus, these companies are characterized by being short-lived, as well as being one-idea or one-product oriented.

2. UNSTRUCTURED GRID GENERATION

Consider the task of generating an arbitrary unstructured mesh in a given computational domain. The information required to perform this task is:

a) A description of the bounding surfaces of the domain to be discretized;
b) A description of how the element size, shape and orientation should be in space;
c) The choice of element type;
d) The choice of a suitable method to achieve the generation of the desired mesh.

The most common ways to provide these four pieces of information are discussed in the following.

2.1 Description of the Bounding Surfaces of the Domain

S.1 Analytic Functions: This is the preferred choice if a CAD-CAM data base exists for the description of the domain. In this case, Splines, B-Splines or other types of functions are used to describe the surface of the domain. An important characteristic of this approach is that the surface is continuous, i.e. there exist no 'holes' in the information.

S.2 Discrete Data: Here, instead of functions, a cloud of points describes the surface of the computational domain. This choice may be attractive when no CAD-CAM data base exists. Commercial digitizers can gather surface point information at high speeds, (> 30,000 points/sec), allowing a very accurate description of a scaled model or the full configuration [Gd.1]. Notice that this approach leads to a discontinuous surface description. In order not to make any mistakes when discretizing the surface during mesh generation, only the points given in the cloud of points should be selected.

2.2 Variation of Element Size, Shape and Orientation in Space

V.1 Internal Measures of Grid Quality: The idea here is to start from a given surface mesh. After the introduction of a new point or element, the quality of the current grid or front is assessed. Then, a new point or element is introduced in the most critical region. This process is repeated until either a mesh that satisfies a preset measure of quality is achieved [Gv.11], or the number of faces in the front has shrunk to zero [Ga.8]. This technique works well for equilateral elements, requiring minimal user input. On the other hand, it is not very general, as the surface mesh needs to be provided as part of procedure.

V.2 Analytical Functions: In this case, the user codes in a small subroutine the desired variation of element size, shape and orientation in space. Needless to say, this is the least general of all procedures, requiring new coding for every new problem. On the other hand, if the same problem needs to be discretized many times, an optimal discretization may be coded in this way. Although it may seem inappropriate to pursue such an approach within unstructured grids, the reader may be reminded that most current airfoil calculations are carried out using this approach.

V.3 Boxes: If all that is required are regions with uniform mesh sizes, one may define a series of boxes in which the element size is constant. For each location in space, the element size taken is the smallest of all the boxes containing the current location. When used in conjunction with surface definition via quad/octrees, one can automate the point distribution process completely in a very elegant way [Go.1].

V.4 Point/Line/Surface Sources: A more flexible way that combines the smoothness of functions with the generality of boxes or other discrete elements is to define sources. As an example, consider the line source defined by the function:

$$\delta(\mathbf{x}) = \delta_0 \left[1 + \left(\frac{r(\mathbf{x}) - r_0}{r_1}\right)^{\gamma}\right] \quad . \qquad (2.1)$$

The definition of $r(\mathbf{x})$ may be inferred from Figure 2.1. As one can see, a very flexible way of defining mesh spacings is obtained with the 4 input parameters $\delta_0, r_0, r_1, \gamma$. If one collapses the two points $\mathbf{x}_2 \to \mathbf{x}_1$, a point-source is obtained. It is a simple matter to introduce these sources interactively with the mouse once the surface data is available. Obviously, the number of sources should be kept small ($N_s < 50$) in order not to incur penalties in user set-up time and grid generation time.

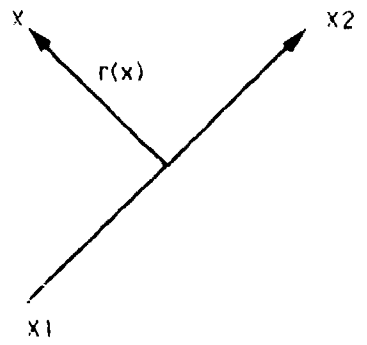


Figure 2.1: Example of a Source to Define Element Size in Space

V.5 Background Grids: Here, a coarse grid is provided by the user. At each of the nodes of this background grid, the element size, stretching and stretching direction are specified. While very general and flexible, and particularly suited to adaptive remeshing, the input of suitable background grids for complex 3-D configurations can become a tedious process. Therefore, most available grid generators employ background grids in conjunction with source-definitions in order to generate the first mesh.

2.3 Element Type

Almost all current unstructured grid generators can only generate triangular or tetrahedral elements. If quad-elements in 2-D are required, they are generated by the following five-stage process:

Q.1 Generate a triangular mesh with elements that are four times as big as the quad-elements required.

Q.2 Fuse as many pairs of triangles into quads as possible without generating quads that are too distorted. This process will leave some triangles in the domain.

Q.3 Smooth the mesh of triangles and quads.

Q.4 H-refine globally the mesh of triangles and quads. For the triangles, introduce an additional point in the element (see Figure 2.2). In this way, the resulting mesh will only contain quads. Moreover, the quads will now be of the desired size.

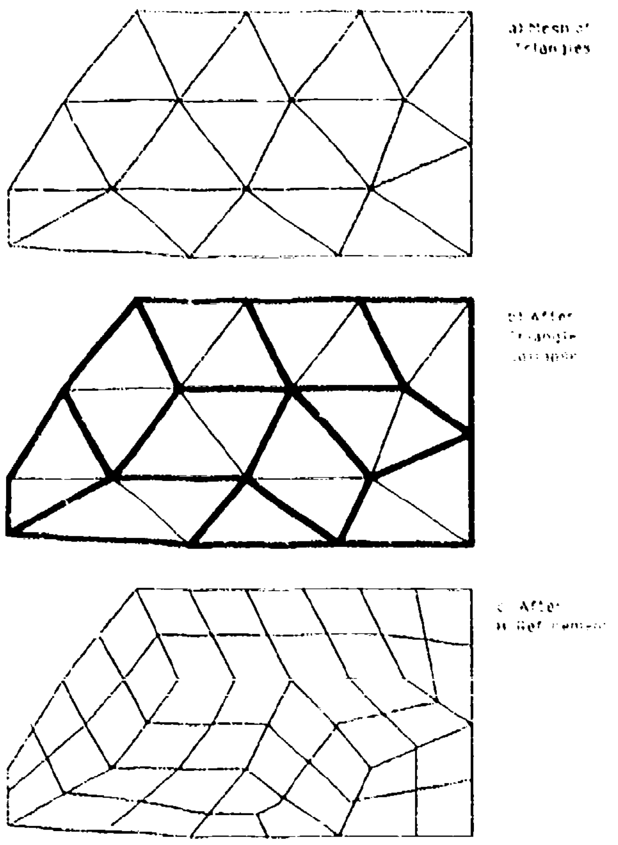


Figure 2.2: Generation of Quad-Meshes from Triangles

Q.5 Smooth the final mesh of quads.

The procedure outlined above will not work in 3-D. There is a large effort of 3-D brick generation technology at Sandia Labs at the present time.

2.4 Methods

There appear to be only the two following ways to fill space with an unstructured mesh:

M.1 Fill Empty, i.e. Not Yet Gridded Space: The idea here is to procede into as yet ungridded space until the complete computational domain is filled with elements. This has been shown diagramatically in Figure 2.3. The methods falling under this category are the so-called advancing front algorithms. The 'front' denotes the boundary between the region in space that has been filled with elements and that which is empty.

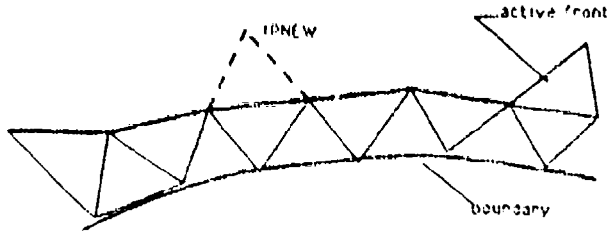


Figure 2.3: Advancing Front Method

M.2 Improve an Existing Grid: In this case, an existing grid is modified by the introduction of new points. After the introduction of each point, the grid is reconnected or reconstructed locally in order to improve the mesh quality. This procedure has been sketched in Figure 2.4. In most cases, the Delauney circumscircle or circumsphere criterion is used to reconnect the points. Given the duality between Voronoi tesselations and the triangulations obtained using the Delauney criterion, the methods falling under this category have been called Voronoi algorithms.

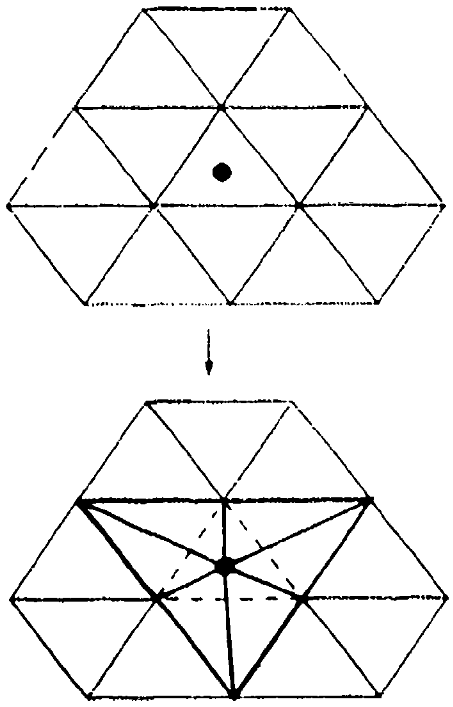


Figure 2.4: Voronoi Approach

In the following matrix, a brief summary is given of possible combinations for specifying element size and shape in space, the method employed to generate the mesh as attempted by various authors:

Authors	Elem Siz	Method	Reference
van Phai	V 2	M 1	Ga 1
Lo	V 2/3	M 1	Ga 2
Morgan/Peraire/Lohner	V 4/5	M 1	Ga 3-7
Huet	V 1	M 1	Ga 8
Cavendish	V 3	M 2	Gv 1
Weatherhill/Mavriplis/Baker	V 2	M 2	Gv 10
Baker	V 3	M 2	Gv 12
Yerry/Shephard	V 3	M 2	Go 1
Holmes	V 1	M 2	Gv 11

2.4 Other Methods

Many other methods of generating meshes that are specially suited to a particular application may be developed. If one knows the approximate answer (say we want to solve the same wing time and time again), the specialized development of an optimal mesh makes good sense. In many of these cases (e.g. an O-mesh for a subsonic/transonic steady-state airfoil calculation), grids generated by the more general methods listed above will tend to be larger than the specialized ones for the same final accuracy. The main methods falling under this specialized category are:

a) Simple Mappings: In this case, it is assumed that the complete computational domain can be mapped into a single quad or cube. The distribution of points and elements in space is controlled either by an algebraic function, or by the solution of a Partial Differential Equation in the transformed space. Needless to say, the number of points on opposing faces of the mapped quad or cube have to match line by line.

b) Macro-Element Approach: Here, the previous approach is applied on a local level by first manually or semi-manually discretizing the domain with large elements. These large elements are subsequently divided up into smaller elements using simple mappings [Gm.1] (see Figure 2.5). In the aerospace community, this approach has been termed 'multi-block', and a whole service industry dedicated to the proper construction of grids has evolved [Gm.2-5].

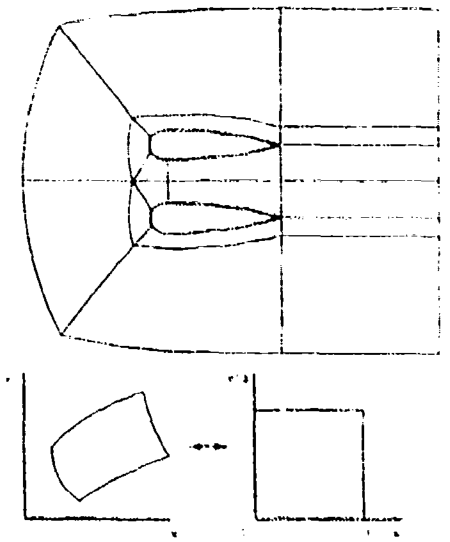


Figure 2.5: Macro-Element or Multi-Block Approach

c) Uniform Background Grid: For problems that require uniform grids (e.g. Radar Cross-Section calculations and homogeneous turbulence), most of the mesh covering the computational domain can be readily obtain from a uniform grid. This grid is called a background grid, because it is laid over the computational domain. At the boundaries, the mesh may be modified to conform to the surfaces, although many Legoland codes omit even this. After modifying these surface elements, the mesh is smoothed, in order to obtain a more uniform discretization close to the boundaries [Gr.1,Gr.2]. The procedure has been sketched in Figure 2.6.

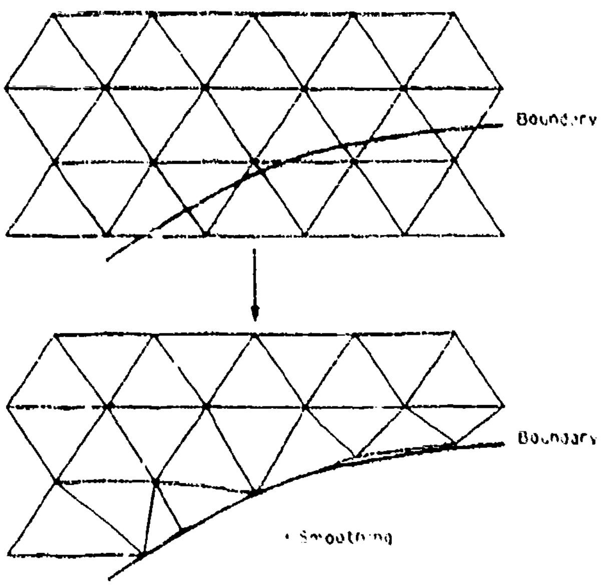


Figure 2.6: Mesh Generation by Perturbation of a Regular Mesh

3. GRID GENERATION USING THE ADVANCING FRONT METHOD

After describing the general strategies currently available to generate unstructured grids, the advancing front method will now be explained in more detail. The aim is to show what it takes to make any of the methods outlined in the previous sections work. Many of the data structures, search algorithms, and general coding issues carry over to the other methods. The advancing front technique consists algorithmically of the following steps:

F.1 Define the spatial variation of element size, stretchings, and stretching directions for the elements to be created. In most cases, this is accomplished with a combination of background grids and sources as outlined above.

F.2 Define the boundaries of the domain to be gridded. This is typically accomplished by splines in 2-D and surface patches in 3-D.

F.3 Using the information stored on the background grid, set up faces on all these boundaries. This yields the initial front of faces. At the same time, find the generation parameters (element size, element stretchings and stretching directions) for these faces from the background grid.

F.4 Select the next face to be deleted from the front; in order to avoid large elements crossing over regions of small elements, the face forming the smallest new element is selected as the next face to be deleted from the list of faces.

F.5 For the face to be deleted:

- F.5.1 Select a 'best point' position for the introduction of a new point IPNEW.
- F.5.2 Determine whether a point exists in the already generated grid that should be used in lieu of the new point. If there is such a point, set this point to IPNEW and continue searching (go to F.5.2).
- F.5.3 Determine whether the element formed with the selected point IPNEW does not cross any given faces. If it does, select a new point as IPNEW and try again (go to F.5.3).

F.6 Add the new element, point, and faces to their respective lists.

F.7 Find the generation parameters for the new faces from the background grid.

F.8 Delete the known faces from the list of faces.

F.9 If there are any faces left in the front, go to F.4.

The complete grid generation of a simple 2-D domain using the advancing front technique is shown in Figure 3.1. In the following, individual aspects of this general algorithmic outlined are described in more detail.

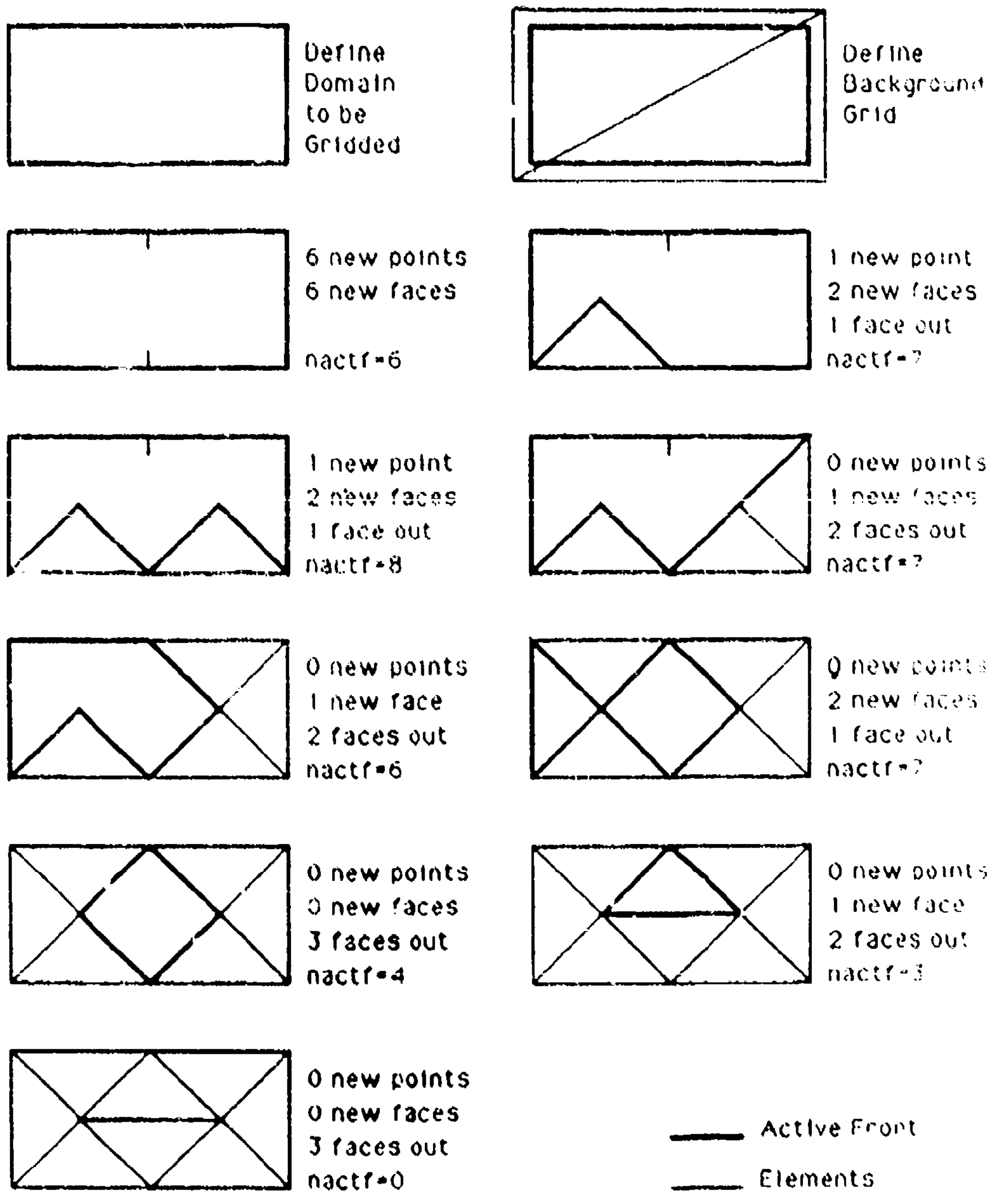


Figure 3.1: Grid Generation Using the Advancing Front Technique

3.1. Checking the Intersection of Faces

The most important ingredient of the advancing front generator is a reliable and fast algorithm for checking whether two faces intersect each other. Experience from practical applications indicates that even slight changes in this portion of the generator greatly influence the final mesh. As with so many other problems in computational geometry, checking whether two faces intersect each other seems trivial for the eye, but is complicated to code. The problem is shown in Figure 3.2. The checking algorithm is based on the following observation: two triangular faces do not intersect if no side of either face intersects the other face. The idea then is to build all possible side-face combinations between any two faces and check them in turn. If no intersection is found, then the faces do not cross. With the notation defined in Figure 3.3, the intersection point is found as

$$\mathbf{x}_f + \alpha^1 \mathbf{g}_1 + \alpha^2 \mathbf{g}_2 = \mathbf{x}_s + \alpha^3 \mathbf{g}_3 \quad , \qquad (3.1)$$

where the $\mathbf{g}_i$-vectors form a covariant basis. Using the contravariant basis $\mathbf{g}^i$ defined by

$$\mathbf{g}^i \cdot \mathbf{g}_j = \delta^i_j \quad , \qquad (3.2)$$

where δ^i_j denotes the Kronecker-delta, the α^i are given by

$$\begin{aligned} \alpha^1 &= (\mathbf{x}_s - \mathbf{x}_f) \cdot \mathbf{g}^1 \quad , \\ \alpha^2 &= (\mathbf{x}_s - \mathbf{x}_f) \cdot \mathbf{g}^2 \quad , \\ \alpha^3 &= (\mathbf{x}_f - \mathbf{x}_s) \cdot \mathbf{g}^3 \quad . \end{aligned} \qquad (3.3)$$

Because we are only interested in a triangular surface for the $\mathbf{g}_1, \mathbf{g}_2$ - plane, we define another quantity similar to the third shape function for a linear triangle:

$$\alpha^4 = 1 - \alpha^1 - \alpha^2 \quad . \qquad (3.4)$$

Using the α^i, two faces can be considered as "crossed" if they only come close together. Then, in order for the side not to cross the face, at least one of the α^i has to satisfy

$$t > max(-\alpha^i, \alpha^i - 1) \quad , i = 1,4 \quad , \qquad (3.5)$$

where t is a predefined tolerance. By projecting the $\mathbf{g}_i$ onto their respective unit contravariant vectors, one can obtain the actual distance between a face and a side. The criterion given by Eqn.(3.5) would then be replaced by (see Figure 3.4):

$$d > \frac{1}{|\mathbf{g}^i|} max(-\alpha^i, \alpha^i - 1) \quad , i = 1,4 \quad . \qquad (3.6)$$

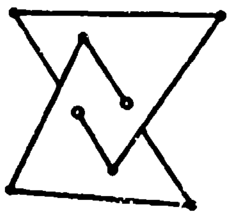


Figure 3.2: Crossing of Two Faces

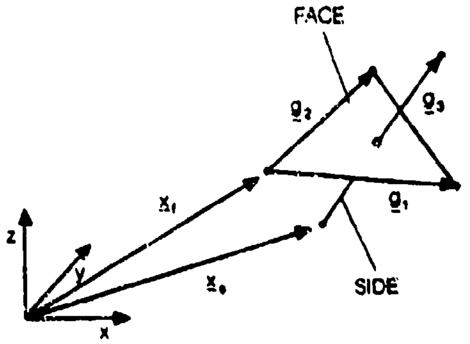


Figure 3.3: Face-Side Combination

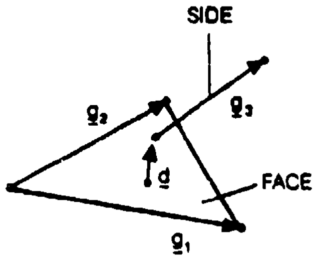


Figure 3.4: Distance Between Face and Side

The first form (Eqn.(3.5)) produces acceptable grids. If the face and the side have points in common, then the α^i will all be either 1 or 0. As both Eqn.(3.5) and Eqn.(3.6) will not be satisfied, special provision has to be made for these cases. For each two faces, six side-face combinations are possible. Considering that on average about 40 close faces need to be checked, this way of checking the crossing of faces is very CPU-intensive. When it was first implemented, this portion of the grid generation code took more than 80% of the CPU time required. In order to reduce the work load, a three-layered approach was subsequently adopted:

a) Min/Max-search: The idea here is to disregard all face-face combinations where the distance between faces exceeds some prescribed minimum distance. This can be accomplished by checking the maximum amd minimum value for the coordinates of each face. Faces cannot possibly cross each other if, at least for one of the dimensions $i = 1,2,3$, they satisfy one of the following inequalities

$$max_{face1}\,(x^i_A, x^i_B, x^i_C) < min_{face2}\,(x^i_A, x^i_B, x^i_C) - d \quad , \qquad (3.7a)$$

$$min_{face1}\,(x^i_A, x^i_B, x^i_C) > max_{face2}\,(x^i_A, x^i_B, x^i_C) + d \quad , \qquad (3.7b)$$

where A, B, C denote the corner points of each face.

b) Local element coordinates: The purpose of checking for face-crossings is to determine whether the newly formed tetrahedron breaks already given faces. The idea is to extend the previous Min/Max-criterion with the shape functions of the new tetrahedron. If all the points of a given face have shape-function values N^i that have the same sign and lie outside the

$[-t,\ 1+t]$ interval, then the tetrahedron cannot possibly cross the face. Such a face is therefore disregarded.

c) In-depth analysis of side-face combinations: All the faces remaining after the filtering process of steps a) and b) are analyzed using side-face combinations as explained above.

Each of these three filters requires about an order of magnitude more CPU-time than the preceding one. When implemented in this way, the face-crossing check requires only 25% of the total grid generation time. When operating on a vector machine, loops are performed over all the possible combinations, building the $\mathbf{g}_i, \mathbf{g}^i, \alpha^i$, etc. in vector mode. Although the vector lengths are rather short, the chaining that results from the lengthy mathematical operations involved results in acceptable megaflop-rates on the CRAY-XMP.

3.2. Data Structures to Minimize Search Overheads

The operations that could potentially reduce the efficiency of the algorithm to $O(N^{1.5})$ or even $O(N^2)$ are (see section 2):

a) Finding the next face to be deleted (step F.4);

b) Finding the closest given points to a new point (step F.5.2);

c) Finding the faces adjacent to a given point (step F.5.3);

d) Finding for any given location the values of generation parameters from the background grid (steps F.3 and F.7). This is an interpolation problem on unstructured grids.

The verb 'find' appears in all of these operations. The main task is to design the best data structures for performing the search operations a)-d) as efficiently as possible. These data structures are typically binary trees or more complex trees. They were developed in the 1960s for Computer Science applications. Many variations are possible (see [Ds.1]). As with flow solvers, there does not seem to be a clearly defined optimal data structure that all current grid generators use. For each of the data structures currently employed, one can find pathological cases where the performance of the tree-seach degrades considerably. The data structures I have used are:

- Heap-lists [Ds.1-4], to find the next face to be deleted from the front;
- Quad-trees (2-D) and Octrees (3-D) [Ds.1,2,Go.1], to locate points that are close to any given location;
- N-trees, to determine which faces are adjacent to a point.

Combining these data-structures, one can also derive an optimal interpolation algorithm for unstructured grids [Ga.4].

3.2.1 Heap List for the Faces

Heap lists are well-known binary tree data structures in computer science [Ds.1-4]. The ordering of the tree is accomplished by requiring that the key of any father (root) be smaller than the keys of the two sons (branches). An example of a tree ordered in this manner is given in Figure 3.5, where a possible tree for the letters of the word 'example' is shown. The letters have been arranged according to their place in the alphabet. We must now devise ways to add or delete entries from such an ordered tree without altering the ordering. In the present case faces have to added as entries into the tree. Therefore, replace 'entry' by 'face'. The ideas that follow use the heap-sort and heap-search algorithms [Ds.2,3] to determine quickly which face should be deleted next from the front.

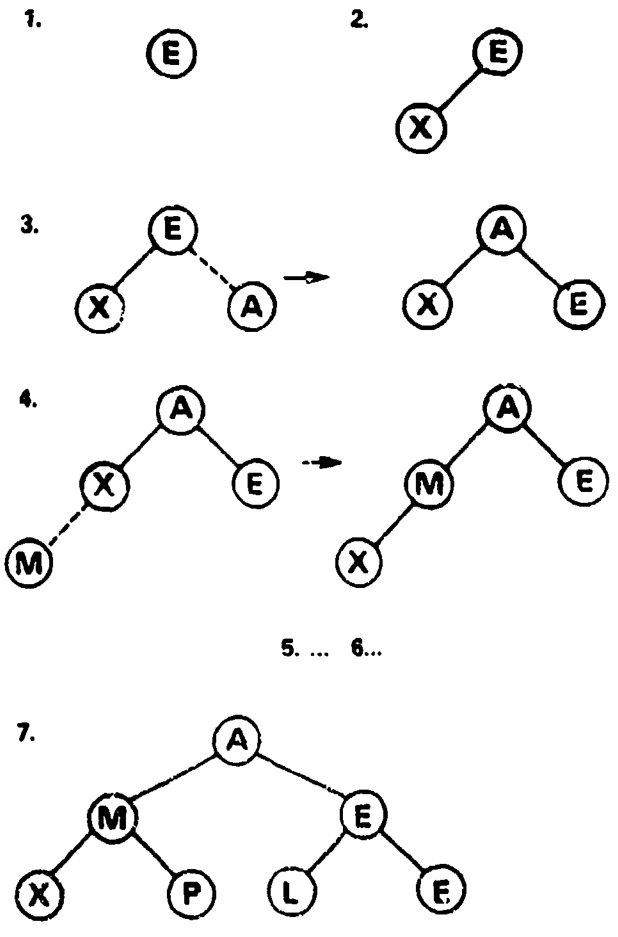


Figure 3.5: Insertion of Items into the Heap List

The positions of the son or the father in the heap list `LHEAP(1:MHEAP)` are denoted by `IPSON`, `IPFATH` respectively. Accordingly, the face-number of the son or the father in the tree is denoted by `IFSON`, `IFFATH`. Then `IFSON=LHEAP(IPSON)` and `IFFATH=LHEAP(IPFATH)`. From Figure 3.5 one can see that the two sons of position `IPFATH` are located at `IPSON1= *IPFATH` and `IPSON2=2*IPFATH+1` respectively. Assume that `NFACE` faces and `RFACE(1:NFACE)` associated keys are given. The two main operations are adding and deleting a new face to the tree without altering the ordering.

3.2.1.1 Adding a new face to the heap list

The idea is to add the new face at the end of the tree. If necessary, the internal order of the tree is re-established by comparing father and son pairs. Thus, the tree is traversed from the bottom upwards.

Algorithm **ADDHEAP** :

A.1 Increase **NHEAP** by one: **NHEAP=NHEAP+1**.

A.2 Place the new face **IFNEW** at the end of the heap list: **LHEAP(NHEAP)=IFNEW**.

A.3 Set the position of the son **IPSON** in the heap list to:
IPSON=NHEAP.

A.4 Then the position of the father **IPFATH** in the heap list is given by: **IPFATH=IPSON/2** (integer division).

A.5 The faces associated with the positions of father and son are: **IFSON=LHEAP(IPSON)**, **IFFATH=LHEAP(IPFATH)**.

A.6 If **RFACE(IFSON) < RFACE(IFFATH)**:

- A.6.1 interchange the faces stored in **LHEAP**
- A.6.2 set **IPSON=IPFAT**.
- A.6.3 unless **IPSON=1** (top of the list), go back to step A.4.

In this way, the face with the smallest associated key **RFACE(IFACE)** remains at the top of the list in position **LHEAP(1)**. The process is illustrated in Figure 3.5, where the letters of the word 'example' ha been inserted sequentially into the heap list.

3.2.1.2 Removing the face at the top of the heap list

The idea is to take out the face at the to of the hea list, replacing it by the face at the bottom of the heap list. If necessary, the internal order [illegible] re-established by comparing pai[illegible] [illegible]ons. Thus, the tree is traversed from [illegible] [illegible]wards.

Algorithm **REMHEAP** :

R.1 Take out the face at the top of the list: **IFOUT=LHEAP(1)**

R.2 Place the face stored at the end of the heap list at the top:
LHEAP(1)=LHEAP(NHEAP), and lower **NHEAP**: **NHEAP=NHEAP-1**.

R.3 Set the position of the father **IPFATH** in the heap list to
IPFATH=1.

R.4 Then the positions of the two sons **IPSON1** and **IPSON2** in the heap list are given by: **IPSON1=2*IPFATH**
and **IPSON2=IPSON1+1**.

R.5 The faces associated with the positions of father and sons are:
IFSON1=LHEAP(IPSON1),
IFSON2=LHEAP(IPSON2),
IFFATH=LHEAP(IPFATH).

R.6 Determine which son needs to be exchanged:
If **RFACE(IFFATH) < RFACE(IFSON1)**, **RFACE(IFSON2)** : set **IPEXCH=0**
If **RFACE(IFFATH) > RFACE(IFSON1) > RFACE(IFSON2)** : set **IPEXCH=IPSON2**
If **RFACE(IFFATH) > RFACE(IFSON2) > RFACE(IFSON1)** : set **IPEXCH=IPSON1**
If **RFACE(IFSON1) > RFACE(IFFATH) > RFACE(IFSON2)** : set **IPEXCH=IPSON2**
If **RFACE(IFSON2) > RFACE(IFFATH) > RFACE(IFSON1)** :
set **IPEXCH=IPSON1**

R.7 Unless **IPEXCH=0**, exchange father and son positions:

- R.7.1 interchange the faces stored in **LHEAP**
- R.7.2 set **IPFATH=IPEXCH**
- R.7.3 unless **2*IPFATH > NHEAP** (bottom of the list), go back to step R.4.

In this way, the face with the smallest associated key will again remain at the top of the list in position **LHEAP(1)**. The described process is illustrated in Figure 3.6, where the successive removal of the smallest element (alphabetically) from the previously constructed heap list is shown.

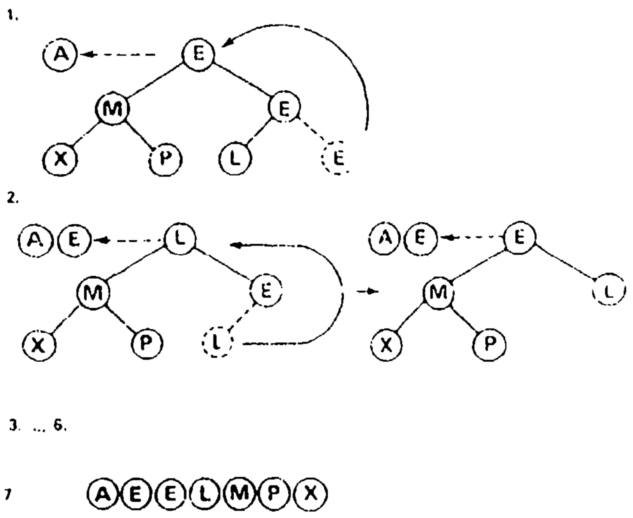


Figure 3.6: Successive Deletion of the Smallest Item from the Heap List

It is easy to prove that both the insertion and the deletion of a face into the heap list will take $O(\log_2$(**NHEAP**)) operations [Ds.3,4] on the average.

[illegible] Quad/Octrees for the Points

Quadtrees and Octrees have been used extensively for 2-D and 3-D grid generators [Go.1]. Their main role there was to define the objects to be gridded, and not to provide an $O(\log(N))$ search algorithm for arbitrary point distributions. The main ideas are described for 2-D regions. The extension to 3-D regions is immediate. Define an array LQUAD(1:7,NQUAD) to store the points, where NQUAD denotes the maximum number of quads allowed. For each quad IQ, store in LQUAD(1:7,IQ) the following information:

LQUAD(7,IQ) :	< 0 :	the quad is full
	$= 0$:	the quad is empty
	> 0 :	the number of points stored in the quad
LQUAD(6,IQ) :	> 0 :	the quad the present quad came from
LQUAD(5,IQ) :	> 0 :	the position in the quad the present quad came from
LQUAD(1:4,IQ) :	for LQUAD(7,IQ) > 0 :	the points stored in this quad
	for LQUAD(7,IQ) < 0 :	the quads into which the present quad was subdivided

At most four points are stored per quad. If a fifth point falls into the quad, the quad is subdivided into four, and the old points are re-located into their respective quads. Then the fifth point is introduced to the new quad into which it falls. If the quad is full again, the subdivision process continues, until a quad with vacant storage space is found. This process is illustrated in Figure 3.7. The newly introduced point E falls into the quad IQ. As IQ already contains the four points A,B,C and D, the quad is subdivided into four. Points A,B,C and D are relocated to the new quads, and point E is added to the new quad NQUAD+2. Figure 3.7 also shows the entries in the LQUAD-array, as well as the associated tree-structure.

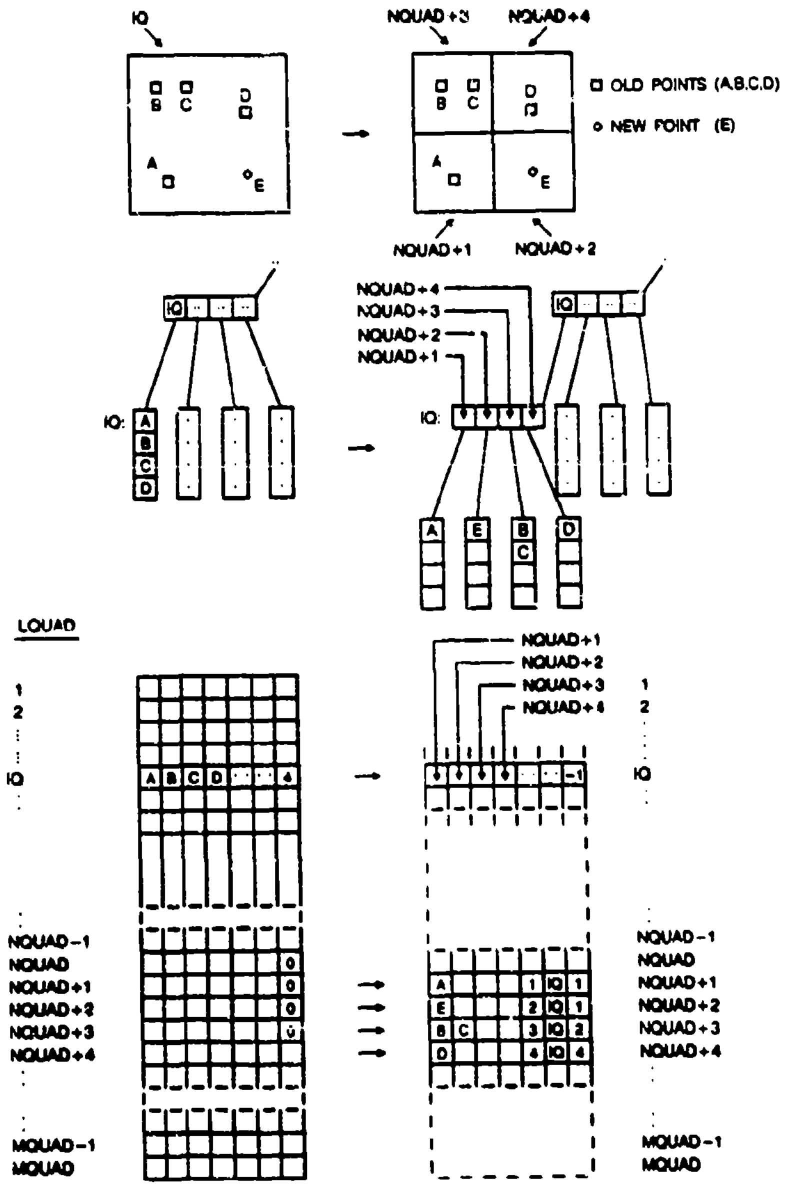


Figure 3.7: Introduction of a New Point into a full Quad

In order to find points that lie inside a search region, the quadtree is traversed from the top downwards. In this way, those quads that lie outside the search region are eliminated at the highest possible level. An attractive feature of quadtrees and octrees is that there is a close relationship between the spatial location and the location in the tree. This considerably reduces the number of operations required to find the quads covering a desired search-region. It is not difficult to see that with the quadtree or the octree it takes $O(\log_4(N))$ or $O(\log_8(N))$ operations to locate all points inside a search region or to find the point closest to a given point.

3.2.3 N-trees for the Face/Point Search

N-trees are linked lists that are often used to relate data of different nature and number of items. For unstructured grids one could relate element, face, side and point-data. In the present case, a storage scheme is sought to answer the question: which are the faces adjacent to a given point? As the number of faces surrounding a point varies from point to point, but usually fluctuates within certain bounds, the following scheme appears attractive. Define an array **LPOIN(1:NPOIN)** over the points and another array **LFAPO(1-MFSUP,MFAPO)**, where **NPOIN** denotes the number of points, **MFSUP** the average number of faces surrounding points (+1), and **MFAPO** the maximum number of storage locations. Then store in:

LPOIN(IPOIN) :		the place **IFAPO** in **LFAPO** where the storage of the faces surrounding point **IPOIN** starts.
LFAPO(MFSUP,IFAPO) :	> 0 :	the number of stored faces
	< 0 :	the place **JFAPO** in **LFAPO** where the storage of the faces surrounding point **IPOIN** is continued
LFAPO(1:MFSUP-1,IFAPO) :	$= 0$:	an empty location
	> 0 :	a face surrounding **IPOIN**

In 2-D one typically has two faces adjacent to a point, and **MFSUP=3**, while for 3-D meshes typical values are **MFSUP=8-10**. Once this storage scheme has been set up, storing and/or finding the faces surrounding points is readily done. The process of adding a face to the linked list **LPOIN/LFAPO** is shown diagramatically in Figure 3.8.

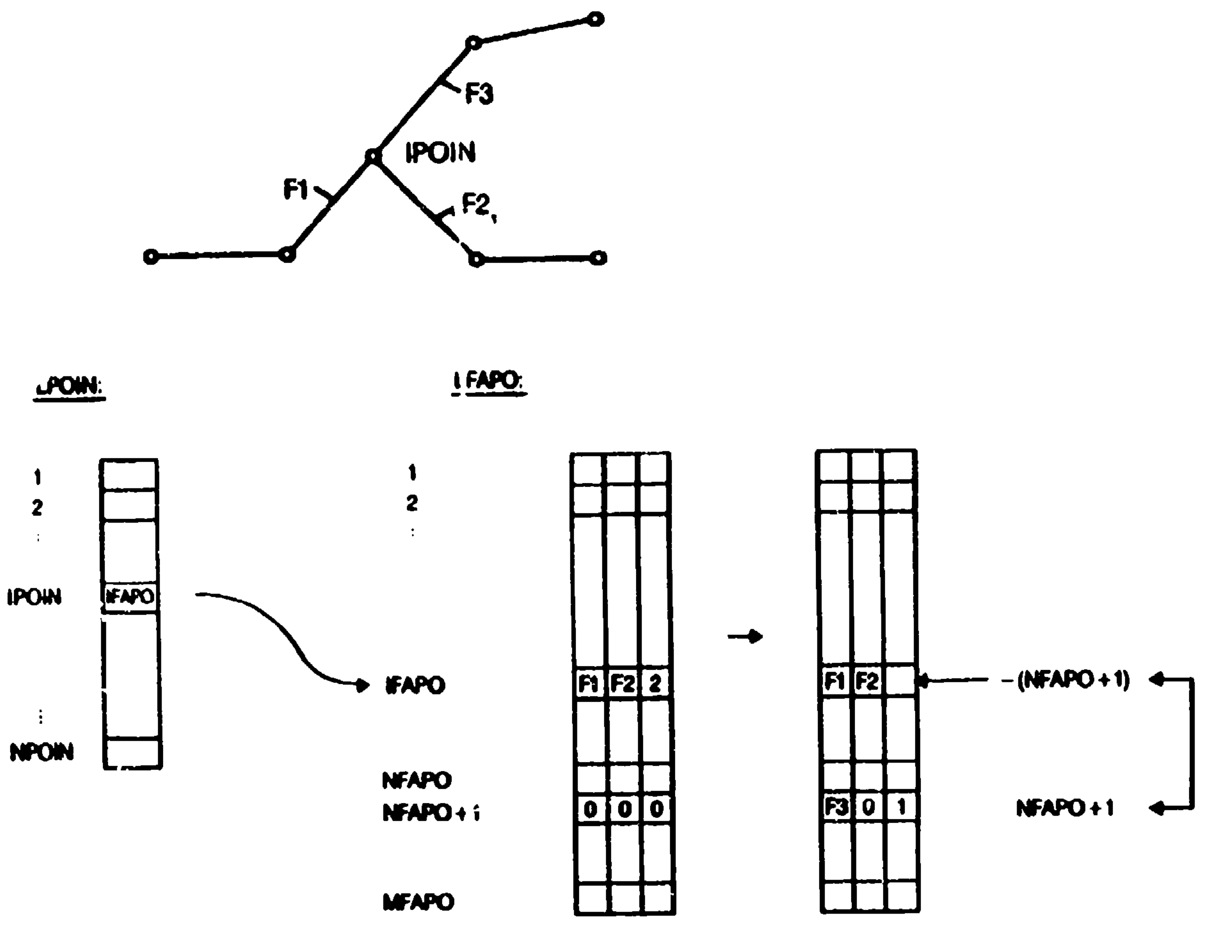


Figure 3.8: Introduction of a New Face into the Linked List

3.3 Additional Techniques to Increase Speed

There are some additional techniques that can be used to improve the performance of the advancing front grid generator. The most important of these are:

a) Filtering: Typically, the number of close points and faces is far too conservative, i.e. large. As an example, consider the search for close points: there may be up to eight points inside an octant, but of these only one may be close to the face to be taken out. The idea is to filter out these 'distant' faces and points in order to avoid extra work afterwards. While the search operations are difficult to vectorize, these filtering operations lend themselves to vectorization in a straightforward way, leading to a considerable overall reduction in CPU requirements.

b) Automatic Reduction of Unused Points: As the front advances into the domain and more and more tetrahedra are generated, the number of tree-levels increases. This automatically implies an increase in CPU-time, as more steps are required to reach the lower levels of the trees. In order to reduce this CPU-increase as much as possible, all trees are automatically restructured. All points which are completely surrounded by tetrahedra are eliminated from the trees. This procedure has proven to be extremely effective. It reduces the asymptotic complexity of the grid generator to less than $O(N \log N)$. In fact, in most practical cases one observes a linear $O(N)$ asymptotic complexity, as CPU is traded between subroutine call overheads and less close faces on average for large problems.

c) Global h-refinement: While the basic advancing front algorithm is a scalar algorithm, h-refinement can be completely vectorized. Therefore, the grid generation process can be made considerably faster by first generating a coarser, but stretched mesh, and then refining globally this first mesh with classic h-refinement [Ae.6,7]. Typical speed-ups achieved by using this approach are 1:6 to 1:7.

3.4 Additional Techniques to Enhance Reliability

The advancing front algorithm described above may still fail for some pathological cases. The newcomer should be reminded that in 3-D, even the slightest chance of something going astray has to be accounted for. In a mesh of over a million tetrahedra (common even for Euler-runs), any slight possibility becomes a reality. The following techniques have been found effective in enhacing the reliability of advancing front grid generators to a point where they can be applied on a routine basis in a production environment:

a) Avoidance of bad elements during generation: It is important not to allow any bad elements to be created during the generation process. These bad elements can cause havoc when trying to introduce further elements at a later stage. Therefore, if a well-shaped tetrahedron can not be introduced for the current face, the face is skipped.

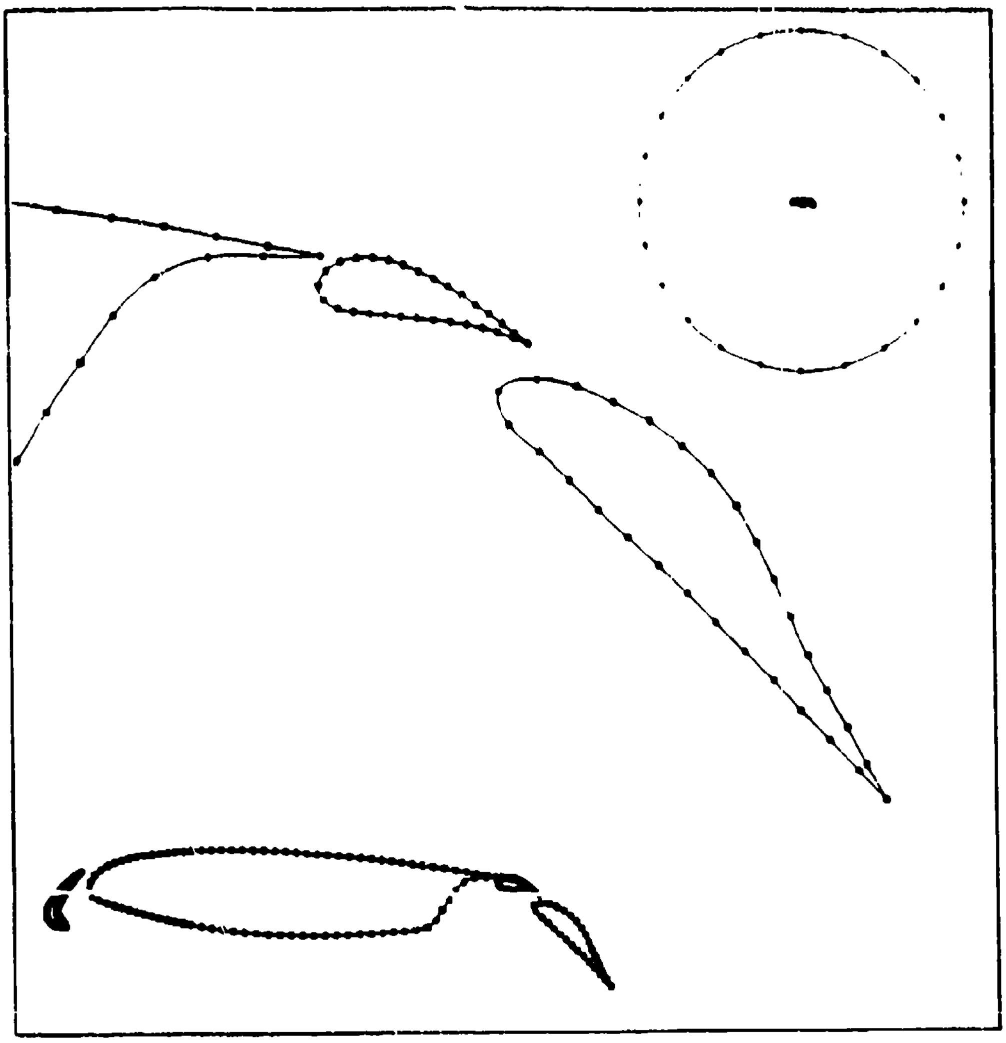
b) Sweep and Retry: If any faces where new elements could not be introduced remain in the field, these regions are enlarged and remeshed again. This 'sweep and retry' technique has proven extremely robust and reliable. It has also made smoothing of meshes possible: if elements with negative or small Jacobians appear during smoothing (as is the case with most spring-analogy smoothers), these elements are removed. The unmeshed regions of space are then regridded. By being able to smooth, the mesh quality is improved substantially, leading to better results in field solvers.

3.5 Some Examples

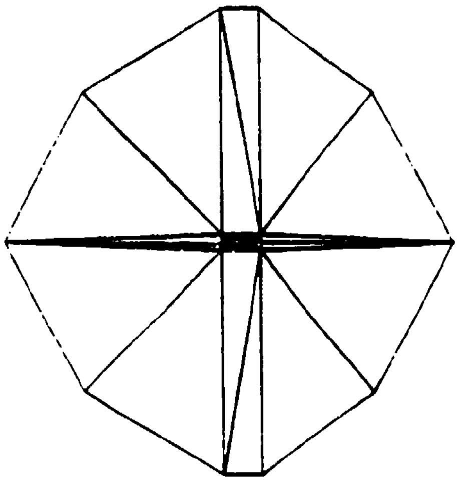
3.5.1 Multi-Element Airfoil Configuration: Figure 3.9 shows a multi-element airfoil case. Figure 3.9a gives the boundary information, Figure 3.9b the background grid used and Figure 3.9c the quadtree obtained for the background grid. Figures 3.9d,e show the constructed mesh before smoothing, as well as the associated quadtree. This grid is then smoothed using a spring system analogy, producing the grid shown in Figure 3.9f.

Currently, the advancing front algorithm constructs grids at a rate of 25,000 tetrahedra per minute on the CRAY-XMP or CRAY-2. With one level of h-refinement, the rate is 190,000 to 200,000 tetrahedra per minute. This rate is essentially independent of grid-size, but may decrease for very small grids.

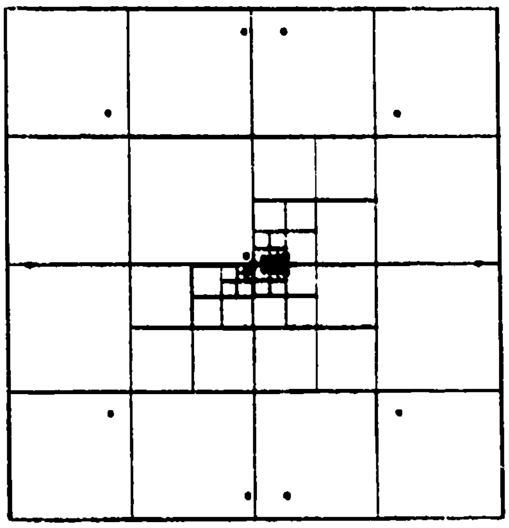
Figure 3.9: Multi-Element Airfoil Configuration



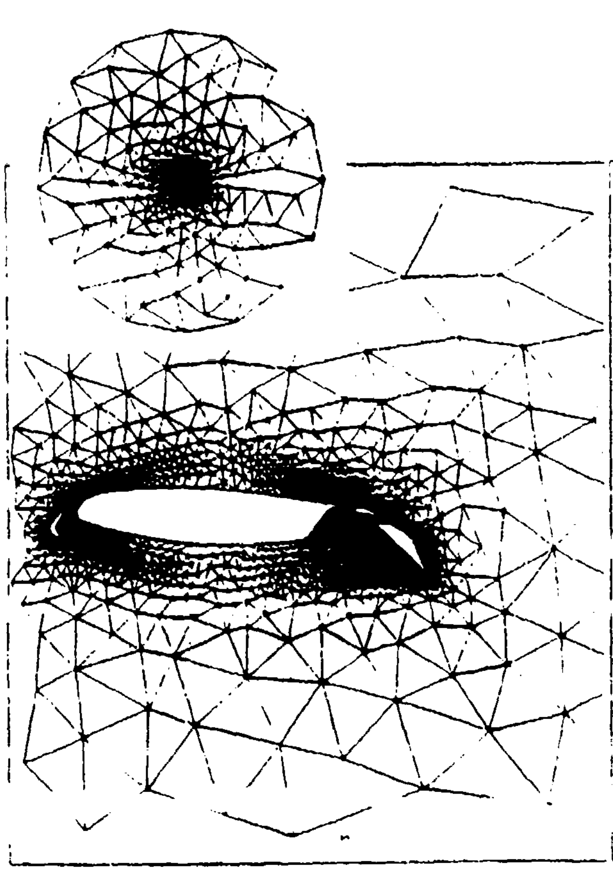
Boundary information
Figure 3.9a



Background grid used
Figure 3.9b

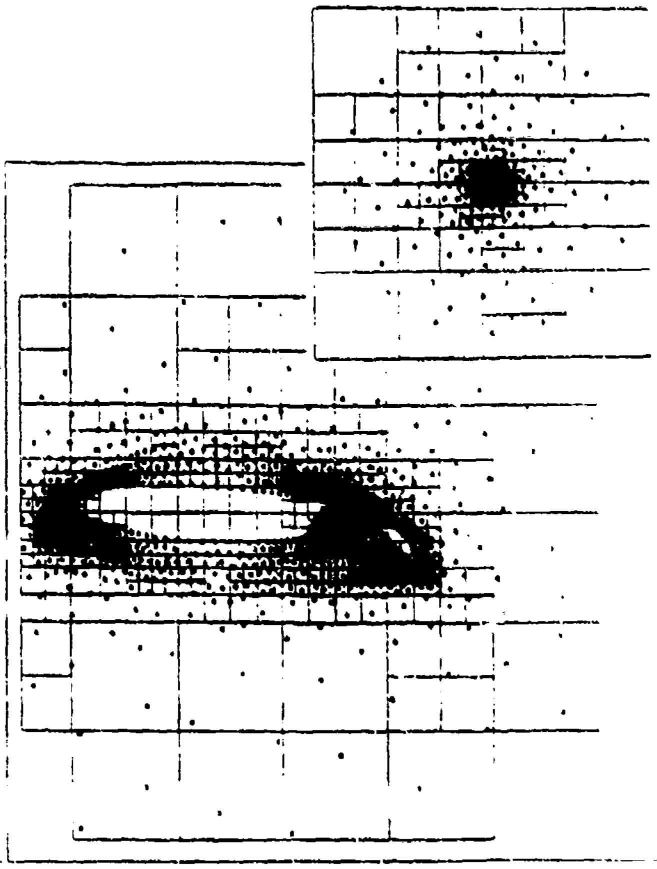


Quadtree of the background grid
Figure 3.9c



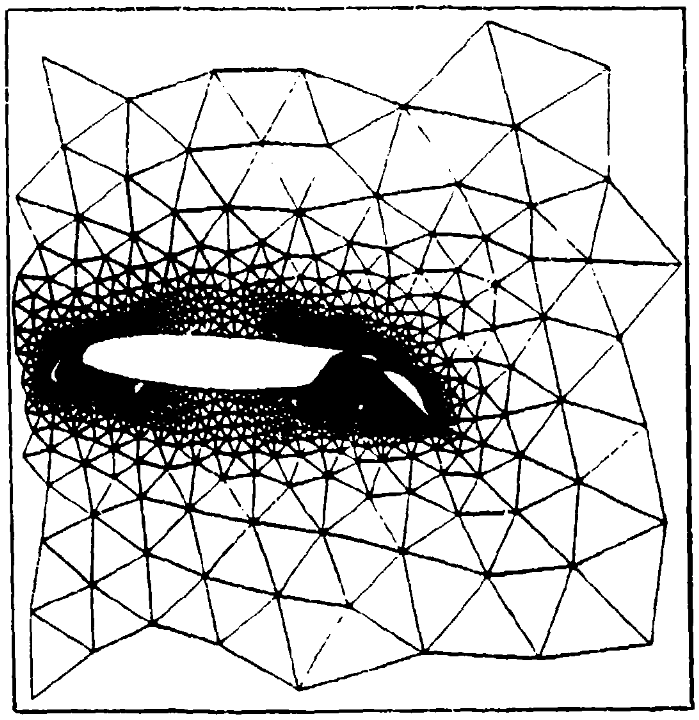
Generated mesh before smoothing

Figures 3.9d



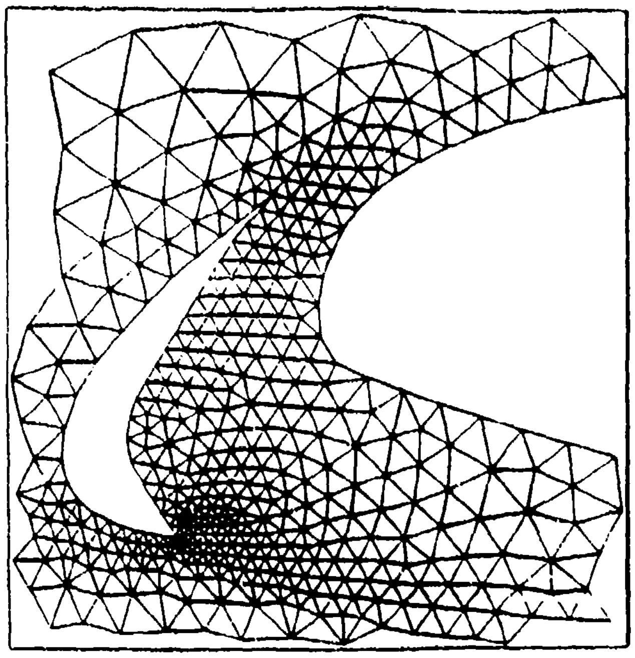
Quadtree of generated mesh

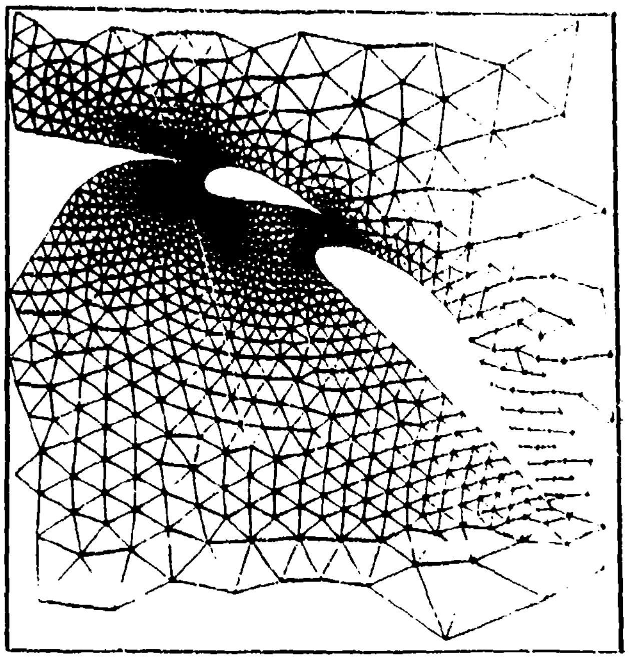
Figure 3.9e

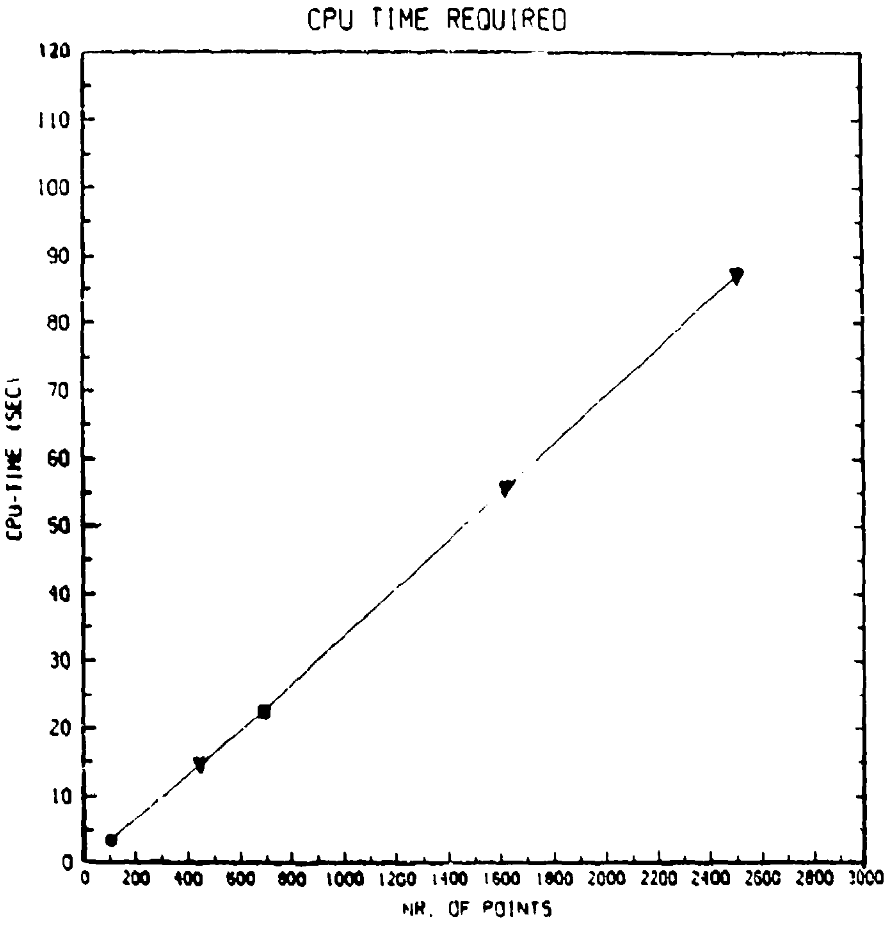


Generated mesh after smoothing

Figure 3.9f.







Timings for several cases

4. ADAPTIVE REFINEMENT

Besides their ability to discretize accurately complex geometries, a second very attractive feature of unstructured grids is the ease with which adaptive refinement can be incorporated into them. The addition of further degrees of freedom does not destroy any previous structure. Thus, the flow solver requires no further modification when operating on an adapted grid. For many practical problems, particularly those with travelling shocks or flapping wakes, the regions that need to be refined are extremely small as compared to the overall domain. Therefore, the savings in storage and CPU-requirements typically range between 10-100, as compared to an overall fine mesh [Am.1-4,Ae.1-12,Ar.1-5]. Experience indicates that for problems falling in this class, adaptive refinement makes the difference between being or not being able to run the problem to an acceptable accuracy in a reasonable time [Ae.8-10]. Without it, one would be forced to use much coarser grids, with lower accuracy, for the same expense.

On the other hand, one should not always expect such big gains from adaptive refinement. As an example, consider the repetitive simulation of inviscid subsonic flow past airfoils. For this application, a well-chosen O-mesh already gives a near-optimal discretization. Therefore, except perhaps at leading and trailing edges and possible shock-regions, adaptation will not give dramatic gains in performance.

The hidden advantage of adaptive refinement is that it frees the user from having to waste time choosing a good initial grid. With adaptation, any initial grid will be transformed into a near-optimal discretization. Thus, adaptation adds a new dimension of user-friendliness to the CFD process that was not there previously.

Any adaptive refinement scheme is composed of three main ingredients. These are

1) an optimal-mesh criterion,
2) an error indicator, and
3) a method to refine and coarsen the mesh.

They give answers to the questions

1) how should the optimal mesh be ?
2) where is refinement/coarsening required ?
3) how should the refinement/coarsening be accomplished ?

The topic of adaptation being now a decade old, it is not surprising that a variety of answers have been proposed by several authors for each of these questions. In the following, the most successful ones are discussed in more depth.

4.1 Optimal Mesh Criteria

The most common optimal mesh criteria employed are:

a) Equidistribution of Error: The aim is to attain a grid in which the error is uniformly distributed in space. One can show that such a mesh has the smallest numbers of degrees of freedom (i.e. the smallest number of elements) for the general aim:

$$\epsilon^h \rightarrow min \; \forall x \in \Omega \; . \qquad (4.1)$$

Conceptually, one can derive this criterion from the observation that the error will have the irregular distribution for the first mesh shown in Figure 4.1a. If the number of degrees of freedom is kept the same, the distribution of element size and shape is all that may be varied. After repositioning of points, the error distribution in space will become more regular, as shown in Figure 4.1b. One can also see that the general aim stated in Eqn.(4.1) will be achieved when the error is constant in the domain.

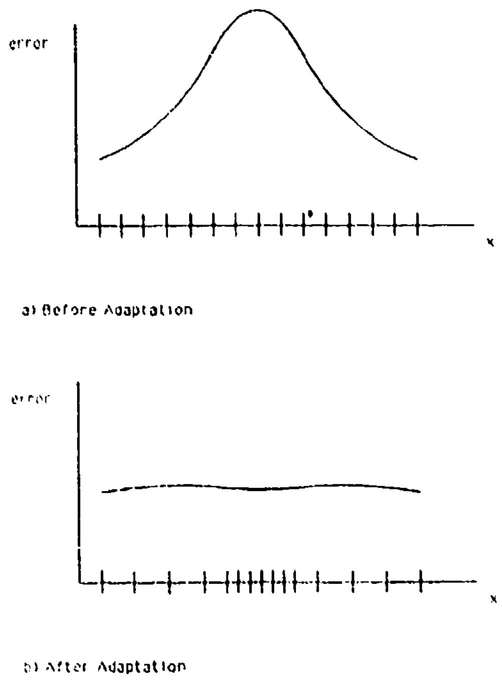


Figure 4.1: Optimal Mesh Criterion

b) Local Absolute Error Tolerances: In many practical applications, the required error tolerances may not be the same at all locations. Moreover, instead of [illegible]ing the general minimization stated in Eqn.(4.1), one may desire to enforce absolute local bounds in certain regions of the domain:

$$\epsilon^h < c_1 \; \forall x \in \Omega_{sub} \; . \qquad (4.2)$$

Mesh refinement or coarsening would then take place if the local error indicator exceeds or falls below given refinement or coarsening tolerances:

$$\epsilon^h > c_r \Rightarrow \text{refine} \;\; , \;\; \epsilon^h < c_c \Rightarrow \text{coarsen.}$$

4.2 Error Indicators/Estimators

Consider the task of trying to determine if the solution obtained on the present mesh is accurate. Intuitively, a number of criteria immediately come to mind: variations of key-variables within elements, entropy-levels, higher-order derivatives of the solutions, etc. All of them make the fundamental assumption that the solution on the present mesh is already in some form 'close' to the exact solution. This assumption is reasonable for parabolic and elliptic problems, where, due to global minimization principles, local deficiencies in the mesh have only a local effect. For hyperbolic problems, the assumption $u^h \approx u$ may be completely erroneous. Consider an airfoil at high angle of attack. A coarse initial mesh may completely miss local separation bubbles at the leading edge that lead to massive separation in the back portion of the airfoil. Thus, any error indicator presently in use (and I really mean any) would miss these features, performing adaptation at the wrong places. On the other hand, the assumption $u^h \approx u$ is a very reasonable one for most initial grids. As a matter of fact, it is not so difficult to attain, particularly if a similar problem has been solved before.

4.2.1 Popular Error Indicators:

The most popular error indicators presently used in production codes may be grouped into the following categories:

I.1 Jumps in Indicator Variables: The simplest error indicator is obtained by simply looking at the jump of some indicator variable like the Mach-number, density, or entropy within an element. The underlying assumption is that in those regions where these jumps are large, more elements are required. This assumptions fails at shocks, where the jump will stay the same no matter how fine the mesh is made. Nevertheless, error indicators of this form have been used in industrial applications with success [Ae.3,Ae.11,12].

I.2 Interpolation Theory: Making the assumption that the solution is smooth, one may approximate the error in each elements by a derivative one order higher than the element shape function. For 1-D, this would result in an error indicator at the element level of the form

$$\epsilon^h_{el} = c_1 h^p \frac{\partial^p u}{\partial x^p} \quad , \tag{4.3}$$

where the $p-th$ derivative is obtained by some recovery procedure, and for linear elements $p = 2$. The total error in the computational domain is the given by

$$\epsilon^h_{\Omega} = c_2 \left[\int h^p \frac{\partial^p u}{\partial x^p} d\Omega \right]^{\frac{1}{2}} \quad . \tag{4.4}$$

I.3 Comparison of Derivatives: Again making the assumption that the solution is smooth, one may compare significant derivatives using schemes of different order. As an example, consider the following two approximations to a second derivative:

$$u_{,xx}|_4 = \frac{1}{h^2}(u_{i-1} - 2u_i + u_{i+1}) - \frac{1}{12}h^2 u_{,IV} \tag{4.5a}$$

$$u_{,xx}|_6 = \frac{1}{12h^2}(-u_{i-2} + 16u_{i-1} - 30u_i + 16u_{i+1} - u_{i+2}) + \frac{1}{90}h^4 u_{,VI} \tag{4.5b}$$

The assumption of smoothness in u would allow a good estimate of the error in the second derivatives from the difference of these two expressions. For unstructured grids, one may recover these derivatives with reconstruction procedures.

I.4 Residuals of PDEs on Adjacent Grids: Assume we have a node-centered scheme to discretize the PDEs at hand. At steady state, the residuals at the nodes will vanish. On the other hand, if the residuals are evaluated at the element level, non-vanishing residuals are observed in the regions that require further refinement. This error indicator has been used extensively in France [Ae.4]. Another possibility is to check locally the effect of higher order shape functions introduced at the element level or at element boundaries. These so-called p-refinement indicators have been used extensively for structural FEM applications [Ae.13,14].

All of these error indicators have been used in practice to guide mesh adaptation procedures. They all work for their respective area of application. It seems that the derivation of a proper error indicator is not a difficult task. The analyst usually knowns how to discern a good solution from a bad one. An error indicator built on this knowledge has to work !

4.2.2 Transient Compressible Flows

Transient Compressible Flows, with their travelling shocks of widely different strengths which need adaptation every 5-10 timesteps require more refined error indicators. Design criteria for these error indicators are:

a) The error indicator should be fast.
b) The error indicator should be dimensionless, so that several 'key variables' can be monitored at the same time.
c) The error indicator should be bounded, so that no further user intervention becomes necessary as the solution evolves.
d) The error indicator should not only mark the regions with strong shocks to be refined, but also weak shocks, contact discontinuities and other 'weak features' in the flow.

All of the popular error indicators described above are not dimensionless. This implies that strong shocks produce large error indicators, whereas weak shocks or contact discontinuities produce small ones. Thus, in the end, only the strong shocks would be refined, losing the weak features of the flow. An error indicator that meets the design criteria a)-d) was proposed in [Ae.6]. In general terms, it is of the form

$$error = \frac{h^2\ |second\ derivatives|}{h\ |first\ derivatives| + \epsilon\ |mean\ value|} \quad (4.6)$$

By dividing the second derivatives by the absolute value of the first derivatives the error indicator becomes bounded, dimensionless, and the 'eating up' effect of strong shocks is avoided. The terms following ϵ are added as a 'noise' filter in order not to refine 'wiggles' or 'ripples' which may appear due to loss of monotonicity. The value for ϵ thus depends on the algorithm chosen to solve the PDEs describing the physical process at hand. The multidimensional form of this error indicator is given by

$$E^I = \sqrt{\frac{\sum_{k,l}(\int_\Omega N^I_{,k} N^J_{,l} d\Omega \cdot U_J)^2}{\sum_{k,l}(\int_\Omega |N^I_{,k}| \left[|N^J_{,l} U_J| + \epsilon\left(|N^J_{,l}||U_J|\right)\right] d\Omega)^2}}, \quad (4.7)$$

where N^I denotes the shape-function of node I. This error indicator has performed very well in 2-D over the years [Ae.6-10,Ar.2-5]. However, when first used in 3-D, it proved unreliable. The source for this seemingly inconsistent behaviour was found to stem from the large local variations in element size, shape, as well as number of element surrounding a point encountered in typical 3-D unstructured grids. These will produce large variations of the second term in the denominator which are not based on physics, but on the mesh structure itself. The solution was to modify this error indicator as follows:

$$E^I = \sqrt{\frac{\sum_{k,l}(\int_\Omega N^I_{,k} N^J_{,l} d\Omega \cdot U_J)^2}{\sum_{k,l}(\int_\Omega |N^I_{,k}||N^J_{,l} U_J| d\Omega)^2 + \epsilon M M_I h_I^{-2} |U_I|}}, \quad (4.8)$$

where MM_I is the lumped mass-matrix at point I, and h_I the average element length at point I. This error indicator has proven to be remarkably insensitive to local variations in element size and shape, while still yielding the correct indicator values for physical phenomena of interest. This good performance is attributed to the smoothing effects of two averaging operations working simultaneously: the lumped mass-matrix and the point-lenghts.

4.2.2.1 Determination of Element Sizes

If all that is required is a thresholding for refinement or coarsening, the error indicator given by Eqn.(4.8) is sufficient. On the other hand, one may wish to obtain a more precise estimation of the required element size to meet a certain tolerance (e.g. to use within an adaptive remeshing context). In this case, a more precise analysis is required. Defining the derivatives according to order as:

$$D^0_i = c_n\left(|U_{i+1}| + 2\cdot|U_i| + |U_{i-1}|\right) \quad (4.9a)$$

$$D^1_i = |U_{i+1} - U_i| + |U_i - U_{i-1}| \quad (4.9b)$$

$$D^2_i = |U_{i+1} - 2\cdot U_i + U_{i-1}| \quad (4.9c)$$

the error indicator on the present (old) grid E^{old} is given by:

$$E^{old}_i = \frac{D^2_i}{D^1_i + D^0_i} \quad . \quad (4.10)$$

The reduction of the current element size h^{old} by a fraction ξ to $h^{new} = \xi \cdot h^{old}$ will yield a new error indicator of the form

$$E^{new}_i = \frac{D^2_i \xi^2}{D^1_i \xi + D^0_i} \quad . \quad (4.11)$$

Given the desired error indicator value E^{new} for the improved mesh, the reduction factor ξ is given by:

$$\xi = \frac{E^{new}}{E^{old}}\frac{1}{2}\left[\frac{D^1_i + \sqrt{(D^1_i)^2 + 4D^0_i \frac{E^{old}}{E^{new}}[D^1_i + D^0_i]}}{[D^1_i + D^0_i]}\right] \quad (4.12)$$

Observe that for a smooth solution with $D^1 \ll D^0$, this results in $\xi = (E^{new}/E^{old})^{0.5}$, consistent with the second order accuracy of linear elements. Close to discontinuities $D^1 \gg D^0$, and one obtains $\xi = E^{new}/E^{old}$, consistent with the first order error obtained in these regions.

This error indicator can be generalized to multidimensional situations by defining the following tensors:

$$(D^0)^I_{kl} = h^2 c_n \int_\Omega |N^I_{,k}||N^J_{,l}||U_J| d\Omega \quad , \quad (4.13)$$

$$(D^1)^I_{kl} = h^2 \int_\Omega |N^I_{,k}||N^J_{,l} U_J| d\Omega \quad ,$$

$$(D^2)^I_{kl} = h^2 \left| \int_\Omega N^I_{,k} N^J_{,l} d\Omega\ U_J \right| \quad , \quad (4.14)$$

which yield an error matrix $\mathbf{E}$ of the form:

$$\mathbf{E} = \begin{Bmatrix} E_{xx} & E_{yx} & E_{zx} \\ E_{xy} & E_{yy} & E_{zy} \\ E_{xz} & E_{yz} & E_{zz} \end{Bmatrix} = \mathbf{X} \cdot \begin{Bmatrix} E_{11} & 0 & 0 \\ 0 & E_{22} & 0 \\ 0 & 0 & E_{33} \end{Bmatrix} \cdot \mathbf{X}^{-1} \quad (4.15)$$

The principal eigenvalues of this matrix are then used to obtain reduction parameters $\xi_{j'j'}$ in the three associated eigenvector directions. Due to the symmetry of $\mathbf{E}$, this is an orthogonal system of eigenvectors that defines a local coordinate system.

4.2.2.2 Smoothing of Element Size and Stretchings

As mentioned above, it is very important in the context of transient problems to generate grids which do not exhibit minimum element sizes that are much smaller than the prescribed minimum element size. Practical calculations indicate that the grids produced by simply taking the described error indicator and the resulting distribution of element sizes, stretchings and stretching directions did not meet this requirement. In other words, they were too irregular. Far superior grids were obtained by smoothing and limiting the initial discributions obtained for element sizes, stretchings and stretching directions.

Smoothing of Element Lengths: The element length h is a scalar quantity. Within each smoothing pass over the mesh, the following operations are performed:

Ss.1 For each element el: take the average element length h^{ave}_{el} over its nodes:

$$h^{ave}_{el} = \frac{1}{nnoel}\sum_{i=1}^{nnoel} h_i \quad , \qquad (4.16)$$

where $i = 1, ..., nnoel$ represent the nodes of element el.

Ss.2 For each point i: form the average element length l^{ave}_i over its surrounding elements:

$$l^{ave}_i = \frac{1}{nsuel}\sum_{el=1}^{nsuel} h^{ave}_{el} \quad , \qquad (4.17)$$

where $el = 1, ..., nsuel$ represent the elements surrounding node i.

Ss.3 For each point i: obtain the new element length h_i from

$$h_i = min(l^{ave}_i, h_i) \quad . \qquad (4.18)$$

Note that a 'point average' is taken, and not an 'area weighted average'. This is important, as it limits the infuence of large elements over smal-l ones. One can also observe that the element length is never allowed to increase at any given point.

Smoothing of Stretchings and Stretching Directions: The stretching direction $\mathbf{s}$ is a vector quantity. Therefore, problems may arise when trying to smooth stretching directions. A typical case is shown in Figure 4.2. Suppose an average stretching vector in the element is desired. In order to accentuate the stretching direction with the maximum stretching, we multiply each stretching vector with its corresponding stretching factor. Simple averaging would then yield the completely erroneous element stretching direction denoted by $\mathbf{s}_0$ in Figure 4.2. In order to avoid this problem, the following algorithm is employed:

Sv.1 For each element el: compute the maximum stetching $\mathbf{s}^{max}_{el}$ encountered at the nodes:

$$\mathbf{s}^{max}_{el} = \max_{i=1,nnoel} \mathbf{s}_i \quad . \qquad (4.19)$$

Sv.2 For each element el: form the average element stretching $\mathbf{s}^{av}_{el}$ as

$$\mathbf{s}^{av}_{el} = \frac{1}{nnoel}\sum_{i=1}^{nnoel} \mathbf{s}_i \; sign(\mathbf{s}^{max}_{el}\cdot\mathbf{s}_i) \quad . \qquad (4.20)$$

For the example shown in Figure 4.2, this yields the vector denoted as $\mathbf{s}^{ave}_{el}$.

Sv.3 For each point i: compute the maximum stetching $\mathbf{s}^{max}_i$ over its surrounding elements:

$$\mathbf{s}^{max}_i = \max_{el=1,nsuel} \mathbf{s}^{av}_{el} \quad . \qquad (4.21)$$

Sv.4 For each point i: form the average element stretching $\mathbf{s}_i$ over its surrounding elements:

$$\mathbf{s}_i = \frac{1}{nsuel}\sum_{el=1}^{nsuel} \mathbf{s}_{el} \; sign(\mathbf{s}^{max}_i\cdot\mathbf{s}_{el}) \quad . \qquad (4.22)$$

As before, a point average proves more effective than an area weighted average, as it limits the infuence of large elements over small ones. For typical runs, three to four smoothing passes over the mesh are perfomed for the element lengths and stretchings.

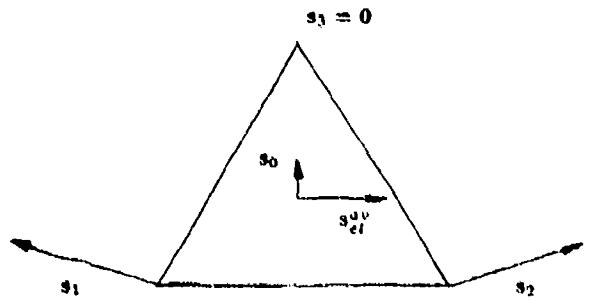


Figure 4.2: Smoothing of Stretchings

4.3 Refinement Strategies

Besides the aim of refinement and the error indicator/estimator, the third ingredient of any adaptive refinement method is the refinement strategy, i.e. how to refine a given mesh. Three different families of refinement strategies have been proposed to date:

S.1 Mesh Movement or Repositioning (r-methods): The aim is to reposition the points in the field in order to obtain a better discretization for the problem at hand. The regions where more elements are required tend to draw points and elements from regions where a coarser mesh can be tolerated. Two basic approaches have been used to date:

a) Spring Systems, whereby the mesh is viewed as a system of springs whose stiffness is proportional to the error indicator, and
b) The Moving Finite Element Method, where the position of points is viewed as further unknowns in a general functional to be minimized.

Mesh movement schemes are relatively simple to code, as the mesh topology is not allowed to change. On the other hand, they are not flexible and general enough for complex production runs which may have many shocks. They have been mentioned here for completeness, and are not recommended for practical applications.

S.2 Mesh Enrichment (h/p-methods): In this case, degrees of freedom are added or taken from a mesh according to some rule. One may either split elements into new ones (h-refinement), or add further degrees of freedom with hierarchical shape-functions (Figure 4.3). The same may be accomplished with the addition of higher order shape-functions (p-refinement), again either conventional ones or hierarchical ones. For elliptic systems of PDEs, the combination of h- and p-refinement leads to exponential convergence rates [Ae.14].

S.3 Remeshing (m-methods): Finally, one may use an advanced unstructured grid generator in combination with an error indicator to remesh the computational domain either globally or locally, in order to produce a more suitable discretization.

4.3.1 Transient Compressible Flows

As before, if we consider transient compressible flows with travelling shocks of widely different strengths which require adaptation every 5-10 timesteps, we are faced with more constraints than the usual steady-state flow application. Design criteria for refinement strategies for these types of applications are:

a) The method should be conservative, i.e. a mesh change should not result in the production or loss of mass, momentum or energy.
b) The method should have a minimal amount of numerical dissipation, as many mesh changes are required during the course of one simulation.
c) The method should not produce elements that are too small, as this would reduce too severely the allowable timestep of the explicit flow solvers employed.
d) The method should be fast. In particular, it should lend itself to a high degree of parallelism.
e) The method should not involve a major storage overhead.

These criteria severely limit the field of applicable strategies. From the results reported in the literature, one can observe that only the simplest and fastest of all refinement strategies, i.e. simple h-refinement with only one level of refinement/coarsening per mesh change, has ever made it into a production environment. The reasons are obvious:

h.1) Conservation presents no problem for h-refinement.
h.2) No interpolations other than the ones naturally given by the element shape-functions are required. Therefore, no numerical diffusion is introduced by the adaptive refinement procedure. This is contrast to adaptive remeshing, where the grids before and after a mesh change may not have the same points in common. The required interpolations of the unknowns will result in an increased amount of numerical diffusion (see [Ar.2]).
h.3) H-refinement is very well suited to vector and parallel processors. This is of particular importance in the present context, where a mesh change is performed every 5-10 timesteps, and a large percentage of mesh points is affected in each mesh chage.
h.4) H-refinement is more robust than remeshing. Particularly in 3-D, the amount of things that can go wrong seems to be much less than when remeshing.

I have tried other methods for this class of problems, like remeshing. Many problems can be solved success-

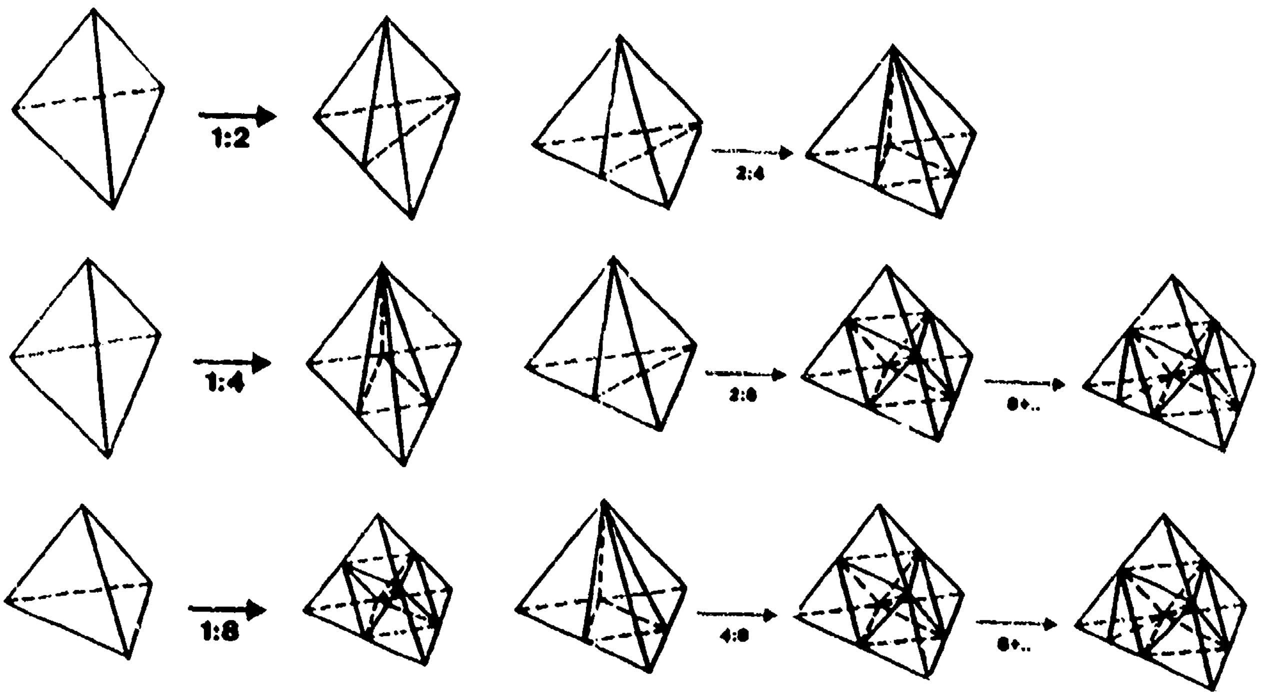


Figure 4.3: Allowable Refinement Cases for Tetrahedra

fully by them, but experience indicates that one can never be sure. Particularly if numerical diffusion due to reinterpolation is present, contact discontinuities may simply disappear, triple points may become distorted, and jets may spread out beyond recognition.

4.4 H-refinement with Tetrahedra

Because of its importance in practical calculations, and as a tutorial example of what is typically required for an h-refinement strategy, the h-refinement with tetrahedra is described in more detail. As stated above, the number of refinement/coarsening levels per mesh change is limited to one. Moreover, refinement of a tetrahedron is only allowed into two (along a side), four (along a face) or eight new tetrahedra. These cases are denoted as 1:2, 1:4 and 1:8 respectively. At the same time, a 1:2 or 1:4 tetrahedron can only be refined further to a 1:4 tetrahedron, or by first going back to a 1:8 tetrahedron with subsequent further refinement of the 8 sub-elements. We call these the 2:4, 2:8+ and 4:8+ refinement cases. The refinement cases are summarized in Figure 4.4. This restrictive set of refinement rules is necessary to avoid the appearance of ill-deformed elements. At the same time, it considerably simplifies the refinement/ coarsening logic. An interesting phenomenon that does not appear in 2-D is the apparently free choice of the inner diagonal for the 1:8 refinement case. As shown in Figure 4.5, one can place the inner four elements around the inner diagonals 5-10, 6-8, or 7-9. In the present case, the shortest inner diagonal was chosen. This choice produces the smallest amount of distorted tetrahedra in the refined grid. When coarsening, again only a limited number of cases that are compatible with the refinement is allowed. Thus, the coarsening cases become 8:4, 8:2, 8:1, 4:2, 4:1, 2:1. These coarsening cases are summarized in Figure 4.6. When constructing the algorithm to refine or coarsen the grid one faces the usual decision of speed versus storage. The more information from the previous grid is stored, the faster the new grid may be constructed. In the present case, this was accomplished by a

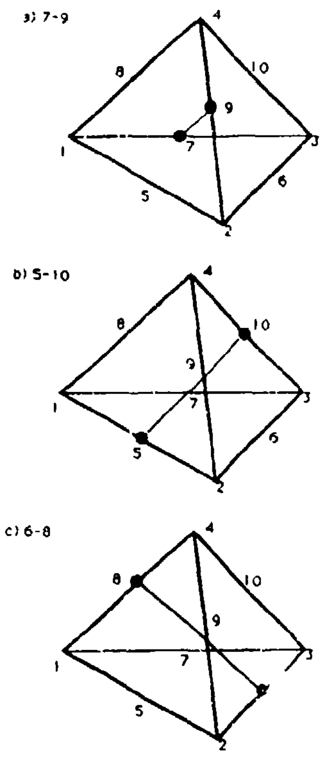


Figure 4.4: Possible Choices for the Inner Diagonals

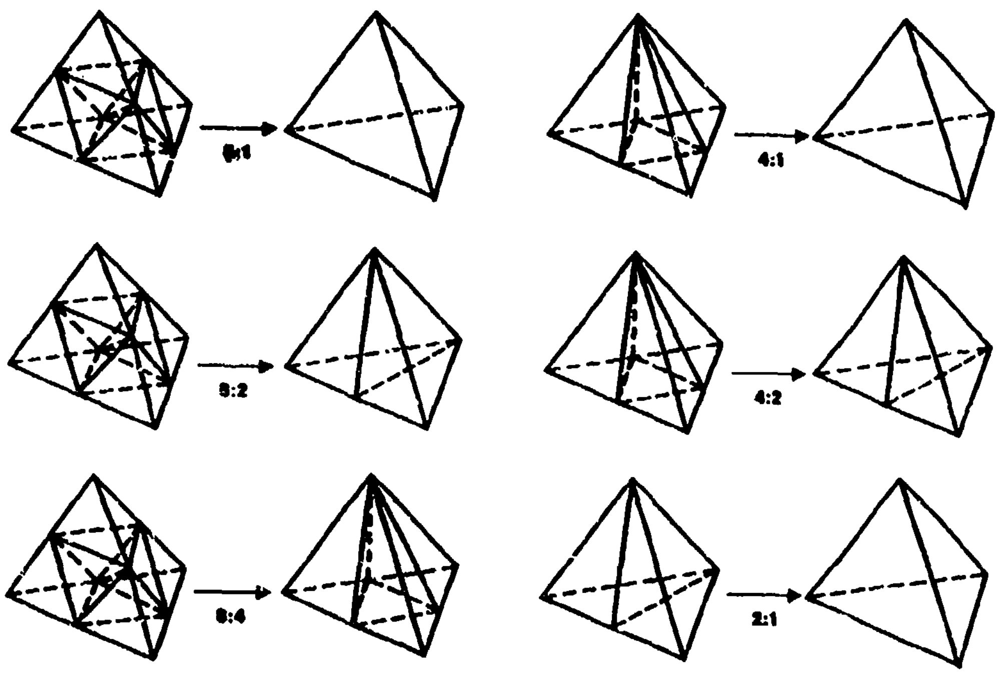


Figure 4.5: Allowable De-Refinement Cases for Tetrahedra

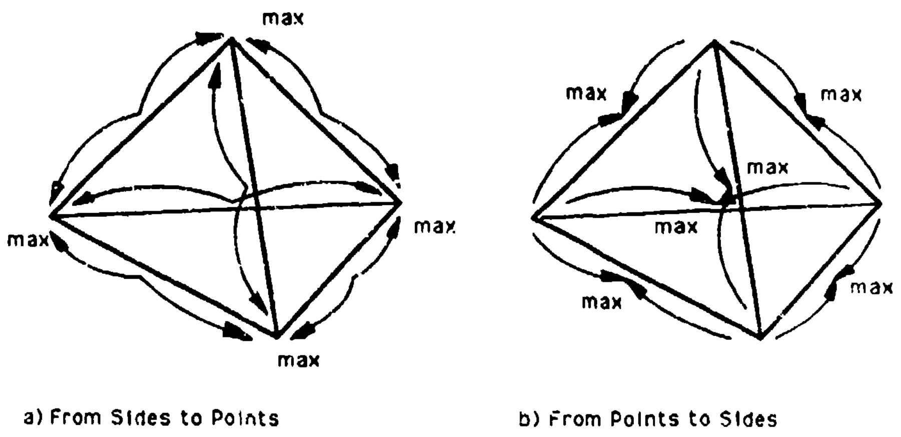


Figure 4.6: Algorithm to Screen for Admissible Refinement Cases

modified tree-structure which requires twelve integer locations per element in order to identify the 'parent' and 'son' elements of any element, as well as the element type.

The first seven integers store the new elements ('sons') of an element that has been subdivided into eight (1:8). For the 1:4 and 1:2 cases, the sons are also stored in this allocated space, and the remaining integer locations are set to zero.

In the eigth integer, the element from which the present element originated (the 'parent' element) is stored.

The ninth integer denotes the position number in the parent element from which this element came.

The tenth integer denotes the element type. One can either have parents or sons of 1:8, 1:4 or 1:2 tetrahedra. These are marked by a positive value of the element type for the parents, and a negative value for the sons. Thus, for example, the son of a 1:8 element would be marked as -8.

Finally, in the eleventh and twelveth integer location, the local and global refinement levels are remembered.

These twelve integer locations per element are sufficient to construct further refinements or to reconstruct the original grid. It is clear that in these twelve integers a certain degree of redundancy is present. For example, the information stored in the 10th integer could be recovered from the data stored in locations 1:8 and 11:12. However, this would require a number of non-vectorizable loops with many IF-tests. Therefore, it was decided to store this value at the time of creation of new elements instead of recomputing it at a later time. Similarly, the 11th integer can be recovered from the information stored in locations 1:8 and 12. As is the case with the 10th integer, storage was traded for CPU-time.

4.4.1 Algorithmic Implementation

Having outlined the basic refinement/coarsening strategy, its algorithmic implementation can now be described in more depth. One complete grid change requires algorithmically the following five steps:

1) Construction of the missing grid information needed for a mesh change (basically the sides of the mesh and the sides adjoining each element);
2) Identification of the elements to be refined;
3) Identification of the elements to be deleted;
4) Refinement of the grid where needed;
5) Coarsening of the grid where needed.

4.4.1.1 Construction of Missing Grid Information

The missing information consists of the sides of the mesh and the sides belonging to each element. The sides are dynamically stored in two arrays, one containing the two points each side connects and the other one (a pointer-array) containing the lowest side-number reaching out of a point. The formation of these two arrays is accomplished in three main loops over the elements, which are partially vectorizable. After having formed these two side-arrays, a further loop over the elements is performed, identifying which sides belong to each element.

4.4.1.2 Identification of Elements to be Refined

The aim of this sub-step is to determine on which sides further gridpoints need to be introduced, so that the resulting refinement patterns on an element-level belong to the allowed cases listed above, thus producing a compatible, valid new mesh. Five main steps are necessary to achieve this goal:

a) Mark elements that require refinement;
b) Add protective layers of elements to be refined;
c) Avoid elements that become too small, or that have been refined too often;
d) Obtain preliminary list of sides where new points will be introduced;
e) Add further sides to this list until an admissible refinement pattern is achieved.

The first three of these steps are obvious. The last two are explained in more detail.

d) Obtain preliminary list of sides for new points
Given the side/element information obtained in substep 4.4.1.1, one can determine a first set of sides on which new gridpoints need to be introduced. This set of sides is still preliminary, as only certain types of refinement are allowed.

e) Add further sides to achive admissible refinement
The list of sides marked for the introduction of new points is still preliminary at this point. In most cases, it will not lead to an admissible refinement pattern to construct a new mesh. Therefore, further sides are marked for the introduction of new points until an admissible refinement pattern is reached. This is accomplished by looping several times over the elements, checking on an element level whether the set of sides marked can lead to an admissible new set of sub-elements. The algorithm used is based on the observation that the admissible cases are based on the introduction of new points along one side (1:2), three contiguous sides (1:4), or six contiguous sides (1:8). These admissible cases can be obtained from the following element-by-element algorithm (see Figure 4.7):

```
c
c     -----this sub assembles the rhs-contributions at points (rhspo)
c          from the element-rhs (rhsel)
c
c          we assume that: lconn: contains the mesh connectivity
c                          rhsel: contains the element-rhsides
c

       This loop will not vectorize, as rhspo in loop 3000 may be accessed
       several times
c
c     -----set rhspo=0.0
c
       call rfilvc(npoin,rhspo,0.0)
c
c     -----loop over the nodes
c
       do 2000 inode=1,nnode
c
c     -----loop over the elements
c
       do 3000 ielem=1,nelem
       ipoin=lconn(ielem,inode)
       rhspo(ipoin)=rhspo(ipoin)+rhsel(ielem,inode)
 3000 continue
c
 2000 continue

       By renumbering the elements, accesing the same point within each group
       of elements can be avoided; thus, one can force vectorization of loop 3000
c
c     -----set rhspo=0.0
c
       call rfilvc(npoin,rhspo,0.0)
c
c     -----outer loop over the groups of elements
c
       iele1=0
c
       do 1000 igrou=1,ngrou
c
c     -----starting and ending elements of this group
c
       iele0=iele1+1
       iele1=lgrou(igrou)
c
c     -----loop over the nodes
c
       do 2000 inode=1,nnode
c
c     -----loop over the elements of this group
c
cdir$ ivdep
c
       do 3000 ielem=iele0,iele1
       ipoin=lconn(ielem,inode)
       rhspo(ipoin)=rhspo(ipoin)+rhsel(ielem,inode)
 3000 continue
c
 2000 continue
c
c     -----end of outer loop over the groups of elements
c
 1000 continue
```

Figure 4.7

- Set the node-array LNODE(1:4)=0;
- Loop over the sides of the element: if the side has been marked for the introduction of a new point, set LNODE(IP1)=1, LNODE(IP2)=1, where IP1, IP2 are the end-nodes corresponding of this side;
- Loop over the sides of the element: if LNODE(IP1)=1 and LNODE(IP2)=1, mark the side marked for the introduction of a new point.

Practical calculations with several admissible layers of refinement and large grids revealed that sometimes up to 15 passes over the mesh where required to obtain an admissible set of sides. This relatively high number of passes can occur when the mesh exhibits regions were the refinement criterion is just met by the elements. Then, the list of sides originally marked for refinement will be far from an admissible one. In each pass over the mesh, a further 'layer' of elements with admissible sides marked for refinement will be added. Moreover, as an element can be refined in six possible ways, in some cases it may take three passes to go from a 1:2 to a 1:8 case. Thus, the 'front' of elements with an admissible set of sides marked for refinement may advance slowly, resulting in many passes over the mesh. A considerable reduction in CPU is realized by presorting the elements as follows:

- Add up all the sides marked for refinement in an element;
- If 0, 1 or 6 sides were marked: do not consider further;
- If 4 or 5 sides were marked: mark all sides of this element to be refined;
- If 2 or 3 sides were marked: analyze in depth as described above.

This then yields the final set of sides on which new gridpoints are introduced.

4.4.1.3 Identification of Elements to be Deleted

The aim of this sub-step is to determine which points are to be deleted, so that the resulting coarsening patterns on an element-level belong to the allowed cases listed above, thus producing a compatible, valid new mesh. Four main steps are necessary to achieve this goal:

a) Mark elements to be deleted;
b) Filter out elements where father and all sons are to be deleted;
c) Obtain preliminary list of points to be deleted;
d) Delete points from this list until an admissible coarsening pattern is achieved.

The first two of these steps are obvious. The last two are explained in more detail.

c) Obtain preliminary list of points to be deleted
Given the list of parent-elements to be coarsened, one can now determine a preliminary list of points to be deleted. Thus, all the points that would be deleted if all the elements contained in this list were coarsened are marked as 'total deletion points'.

d) Delete points to achieve admissible coarsening
The list of total deletion points obtained in the previous step is only preliminary, as unallowed coarsening cases may appear on an element level. Therefore loops are performed over the elements, deleting all those total deletion points which would result in unallowed coarsening cases for the elements adjoining them. This process is stopped when no incompatible total deletion points are left. As before, this process may be made considerably faster by grouping together and treating differently the parent elements with 0,1,2,3,4,5 or 6 total deletion points.

4.4.1.4 Refinement of the Grid Where Needed

The introduction of further points and elements is performed in two independent steps, which in principle could be performed in parallel.

a) Points: To add further points, the sides marked for refinement in sub-step 4.1.2 are grouped together. For each of these sides a new grid-point will be introduced. The interpolation of the coordinates and unknowns is then performed using the side/point information obtained in sub-step 4.1.1. These new coordinates and unknowns are added to their respective arrays. In the same way new boundary conditions are introduced where required, and the location of new boundary points is adjusted using the CAD-CAM data defining the computational domain.

b) Elements: In order to add further elements, the sides marked for refinement are labelled with their new gridpoint-number. Thereafter, the element/side information obtained in sub-step 4.1.1 above is employed to add the new elements. The elements to be refined are grouped together according to the refinement cases shown in Figure 4.4. Each case is treated in block fashion in a separate subroutine. Perhaps the major breakthrough of the present work was the reduction of the many possible refinement cases to only six. In order to accomplish this, some information for the 2:8+ and the 4:8+ cases is stored ahead in scratch arrays. After these elements have been refined according to the 2:8 and 4:8 cases, their sons are screened for further refinement using this information. All sons that require further refinement are then grouped together as 1:2 or 1:4 cases, and processed in turn.

As the original description of all variables was performed using linear elements, the linear interpolation of the unknowns to the new points will be conservative. However, small conservation losses will occur at curved surfaces. These losses are considered to be both unavoidable and small.

4.4.1.5 Coarsening of the Grid Where Needed

The deletion of points and elements is again performed in two independent steps, which, in principle, could be performed in parallel.

a) Points: The points to be deleted, having been marked in sub-step 4.1.3 above, all that remains to be done is to fill up the voids in the coordinate-, unknown- and boundary condition-arrays by renumbering points and boundary conditions.

b) Elements: The deletion of elements is again performed blockwise, by grouping together all elements corresponding to the coarsening cases shown in Figure 4.6. Thereafter, the elements are also renumbered (in order to fill up the gaps left by the deleted elements), and the point-renumbering is taken into consideration within the connectivity-arrays.

It is clear that the coarsening procedure is non-conservative. However, no physical or numerical problems have ever been observed by using it. This may be explained by the fact that the coarsening is done in those regions where the solution is smooth. Thus, the coarsened grid represents the solution very well, and consequently the conservation losses are small. Moreover, those regions where the maintenance of conservation is important (e.g. discontinuities) are never affected.

4.5 Adaptive Remeshing

Adaptive remeshing is a very competitive adaptation technique for steady-state applications or problems with moving bodies. For the latter class of problems, a fixed mesh structure will, in most cases, lead to badly distorted elements. This means that at least a partial regeneration of the computational domain is required. On the other hand, as the bodies move through the flowfield, the positions of relevant flow features will change. Therefore, in most of the computational domain, a new mesh distribution will be required. The idea is to regenerate the whole computational domain adaptively, taking into consideration the current flowfield solution. Any of the automatic grid generation techniques outlined above may be employed to accomplish this. In my codes I tend to use the advancing front technique [Ar.2-4]. The steps required for one adaptive remeshing are as follows:

R.1 Obtain the error indicator matrix for the gridpoints of the present grid.

R.2 Given the error indicator matrix, get the element size, element stretching and stretching direction for the new grid.

R.3 Using the old grid as the 'background grid', remesh the computational domain using the advancing front technique.

R.4 If further levels of global h-refinement are desired, refine the new grid globally.

R.5 Interpolate the solution from the old grid to the new one.

4.5.1 Local Remeshing

Practical simulations indicate that the appearance of badly distorted elements occurs at a frequency that is much higher than expected from the element size prescribed. Given the relatively high cost of global remeshing, local remeshing in the vicinity of the elements that became too distorted becomes an attractive option. The steps required are as follows:

L.1 Identify the badly distorted elements in the layers that move, writing them into a list `LEREM(1:NEREM)`.

L.2 Add to this list the elements surrounding these badly distorted elements.

L.3 Form 'holes' in the present mesh by:

  L.3.1 Forming a new background mesh with the elements stored in the list `LEREM` .

  L.3.2 Deleting the elements stored in `LEREM` from the current mesh.

  L.3.3 Removing all unused points from the grid thus obtained.

L.4 Recompute the error indicators and new element distribution for the background grid.

L.5 Regrid the 'holes' using the advancing front method.

Typically, only a very small number of elements ($<$ 10) becomes so distorted that a remeshing is required. Thus, local remeshing is a very economical tool that allowes reductions in CPU-requirements by more than 60% for typical runs.

5. EFFECTIVE USE OF SUPERCOMPUTER HARDWARE

However clever an algorithm may be, it has to run efficiently on today's supercomputer hardware. The following section examines the main issues involved for each type of supercomputer currently available, as well as techniques to use this hardware as efficiently as possible. There are three main types of supercomputer currently available. These are:

a) The by now traditional vector machines, which achieve high speeds by splitting the necessary arithmetic operations between subsequent members of a vector-loop,

b) Single Instruction Multiple Data (SIMD) machines, that perform the same arithmetic operation across a large number of low-level processors, and

c) Multiple Instruction Multiple Data (MIMD) machines, that perform the different arithmetic operations across many medium-level processors.

One emerging architecture for future machines is the NIMD-machine, where only a few (N) different processes are carried out over a large number (M) of powerful vector processors. An architecture like this would require the programmer to take into consideration all the individual aspects ecountered in each of the presently available supercomputer architectures.

5.1 Vector Machines

At the beginning of the 1980s, two different classes of vector machines where in use: memory to memory (CYBER-205) and register to register (CRAY) architectures. Due to its greater versatility, the second type of machine is the dominant vector machine at present. Register to register machines prefer chunky loops, high flop to memory access ratios, and low indirect addressing. Unfortunately, most simple flow solvers look just the opposite: they have extremely simple loops, low flop to memory access ratios, and high amount of indirect addressing. Just to illustrate the importance of indirect addressing, the reader may be reminded that even with hardware gather/scatter, it takes the equivalent of 2.5 multiplications to get a number from memory using indirect addressing. Therefore, an important issue when coding for performance on these machines is the reduction of indirect addressing operations required. For tetrahedral elements, the amount of indirect addressing operations required can be approximately halved by going from an element-based data structure to an edge-based data structure. This change of data structure avoids redundant information, leading also to a proportional reduction in CPU and memory requirements [Sc.3-5]. A second important issue is the vectorizability of scatter-add loops. Consider the following element RHS assembly loop:

Algorithms to group the elements into non-conflicting groups fall into the category of colouring schemes [Sc.1,2]. This has been exemplified for the 2-D mesh shown in Figure 5.1. Given that on typical vector-machines one only needs vector lengths that are a few multiples of 64, it is easy to construct very well-balanced element orderings that span the whole mesh, except for one last group with less than 64 elements.

5.2 SIMD Machines

SIMD machines, as exemplified by the Thinking Machines CM-series, operate very efficiently on nearest-neighbour transfer of operations. On the other hand, general data exchange operations, as required for unstructured grids, take a very large amount of time. A simple gather takes the equivalent of 20-60 flops. Several routers or pre-compilers have been devised to alleviate this problem [Sc6]. At the same time, renumbering strategies have been explored [Sc7]. Both approaches combined lead to a significant decrease in indirect adressing overhead. On the other hand, they are useless for more general applications where the mesh topology changes every few timesteps (remeshing, h-refinement, etc.). Therefore, only a few steady-state or fixed mesh transient applications have been extremely successfull on this type of machine [Sc.8-11]. SIMD machines may be compared to memory to memory vector machines: they inherently lack generality, which may lead to their eventual demise.

5.3 MIMD Machines

MIMD machines, as exemplified by the Intel, NCube, Parsytech, etc. hypercubes, consist of fairly powerful processors that are linked together by message passing and synchronization software. In the future, each of these processors will be a vector-processor. This implies that most of the algorithmic considerations discussed for vector machines will carry over to these machines. The main issues when trying to code optimally for this type of supercomputer are:

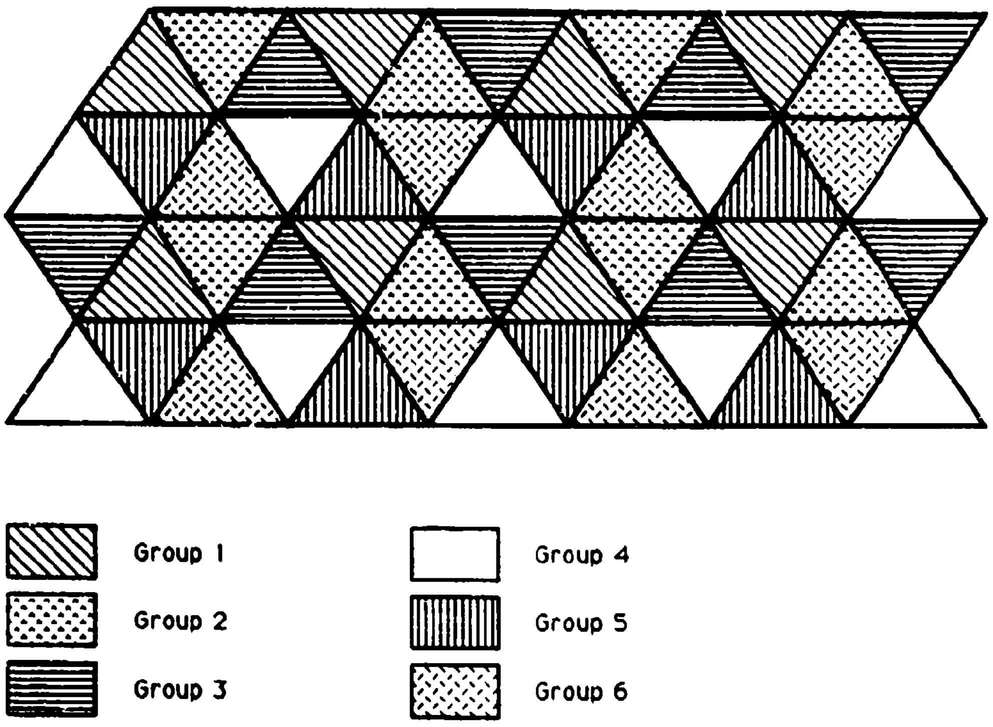


Figure 5.1: Renumbering or Colouring of Elements for Vectorized Scatter-Add

a) Minimization of Idle Time: In the worst case scenario, all but one processor wait for the last processor to finish a certain task. This implies that one has to strive for the same amount of work and communication in each processor.
b) Minimization of Interprocessor Information Flow: With processor performance advancing rapidly in comparison to interprocessor transfer speeds, the amount of information required between processors has to be minimized. In general, this will lead to a minimization problem for the area-to-volume ratio of the processors.
c) Extra Layers/Additional Work Tradeoffs: The amount of required information transfer between processors can be staged (usual code), or more information can be gathered ahead of the timestep. In the latter case, no further information transfer is required within a timestep. On the other hand, more layers of information surrounding each domain are required. Both approaches are possible, and the performance of each may depend more on the indiviual hardware of each machine than anything else.

5.3.1 General Considerations

The effective use of any parallel machine requires the following general steps for the solution of the problem at hand:

P.1 Break up the problem to be solved into pieces;
P.2 Hand each processor a piece of the problem;
P.3 If required: provide for interprocessor transfer of information;
P.4 Assemble the results.

The processor hierarchy and scheduling may also vary. Some of the possible choices are:

- All processors working at the same level (used for grid smoothing and explicit flow solvers);
- A pyramid of masters that hand out and/or perform tasks;
- A one master - many slaves doing different operations hierarchy (used for unstructured grid generation);
- A one master - many slaves doing the same operation hierarchy (the SIMD paradigm).

Porting a typical flow-code to a MIMD requires the following pieces of software:

- Parallel Input Modules;
- Parallel Domain Subdivision Modules for Load Balancing;
- Node-Versions of the Flow-Code for Parallel Execution;
- Interdomain Info-Transfer Modules;
- Parallel Adaptive Grid Regeneration Modules;
- Parallel H-refinement Modules;
- Parallel Output Modules.

The message is clear: do everything in parallel, or don't even start.

5.4. Domain Splitting

The aim of any domain-splitting algorithm devised for parallel machines is to minimize the interprocessor transfer of information. For a given information flux β_i, and subdomains of equal size, this is equivalent to the minimization of the surface to volume ratio of each subdomain i:

$$\frac{\beta_i \Gamma_i}{\Omega_i} \rightarrow min \quad . \qquad (5.1)$$

Several algorithms for splitting a domain can be envisioned. These are, in ascending order of generality (see Figure 5.2):

- Simple Cartesian splitting;
- Quadtree/Octree splitting;
- Unstructured grid splitting.

It seems natural to choose the unstructured grid splitting for the following reasons:

- A fine unstructured grid can be assumed. This, in turn, allows division of the grid into subdomains of nearly equal size.
- Three major pieces of software, the grid smoother, the field solver, and the grid generator can all use the same algorithm to generate subdomains. This reduces the software development costs.

5.4.1 A Domain-Splitting Algorithm

A simple algorithm that attempts to obtain a subdivision which satisfies Eqn.(5.1) is the following 'wavefront'-type scheme:

Assume given; as part of the background grid

- The elements that surround each point;
- An initial starting element within each subdomain.
- A real number REFFE in each element associated with the effort to be spent in it (e.g. the number of elements to be created);
- A real number REFFD that denotes the desired average cumulative effort in each subdomain.

Then:

```
S.1 Initialize point and element arrays;
S.2 Initialize domain counter REFFD ;
S.3 Start next subdomain:
        Update domain number;
        Initialize subdomain effort counter;
S.4 Select the next point to be surrounded from the
    order of creation list;
S.5 Mark this point as totally surrounded;
S.6 For each of the elements surrounding this point:
        If the element has not been marked as be-
        longing to a domain before:
            Mark the element as belonging to
            the present domain;
            Update subdomain effort counter;
            For the nodes of this element:
            If the point is not yet totally
            surrounded and has not
            yet been incorporated into the
            order of creation list:
```

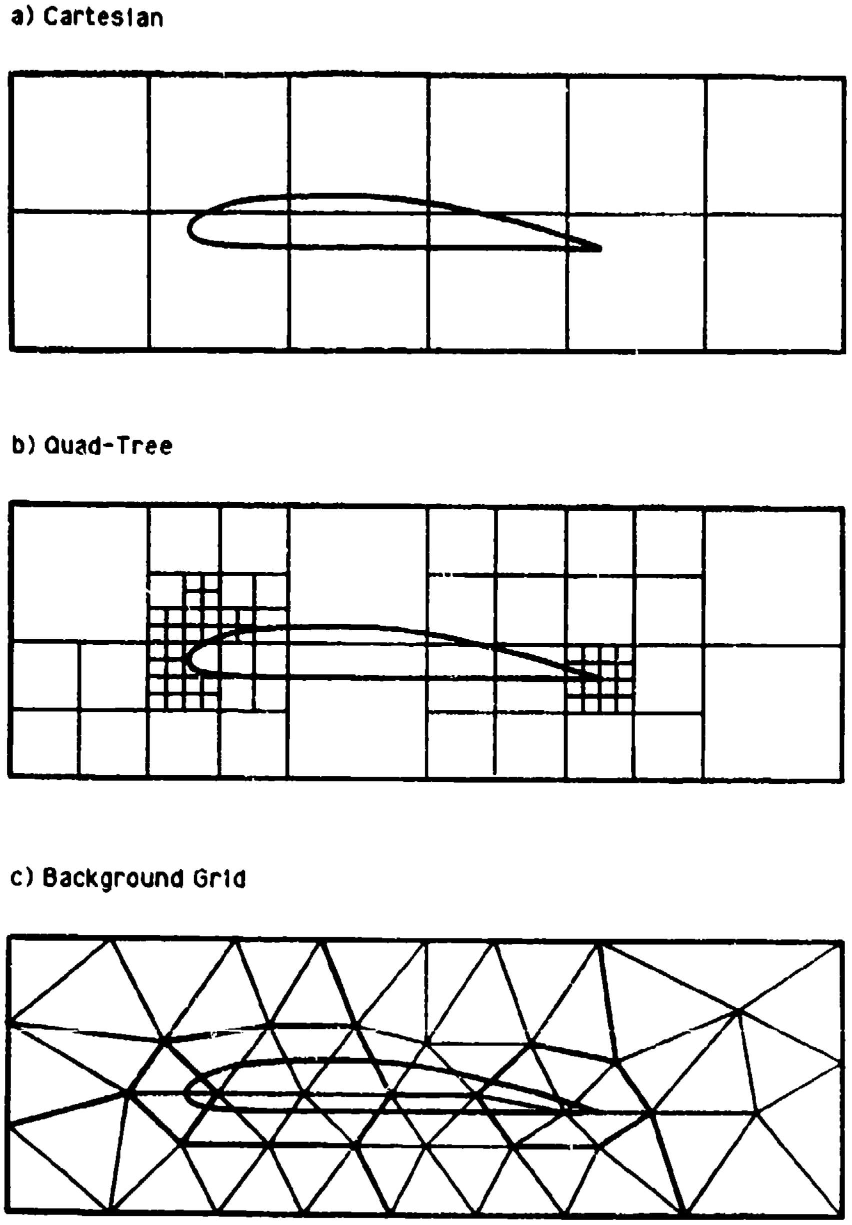


Figure 5.2: Domain Splitting Strategies

```
                Add the point to the order of
                creation list;
            Endif
        Endif
  S.7 If the subdomain effort counter exceeds REFFD :

        Compress point creation list, eliminating all
        totally surrounded points.

Goto S.3
    Otherwise

Goto S.4
    Endif
```

Figure 5.3 shows how this algorithm works on a simple 2-D domain, where it was assumed that REFFE=1.0 and REFFD=8.0. For more expensive, but almost as effective algorithms, see [Mi.1].

5.5 Parallel Grid Generation

For the grid generator, the steps taken are as follows:

G.1 Subdivide the global domain to be gridded into subdomains using the background grid;
G.2 Grid up each subdomain separately;
G.3 Grid up the inter-subdomain regions;
G.4 Assemble the result.

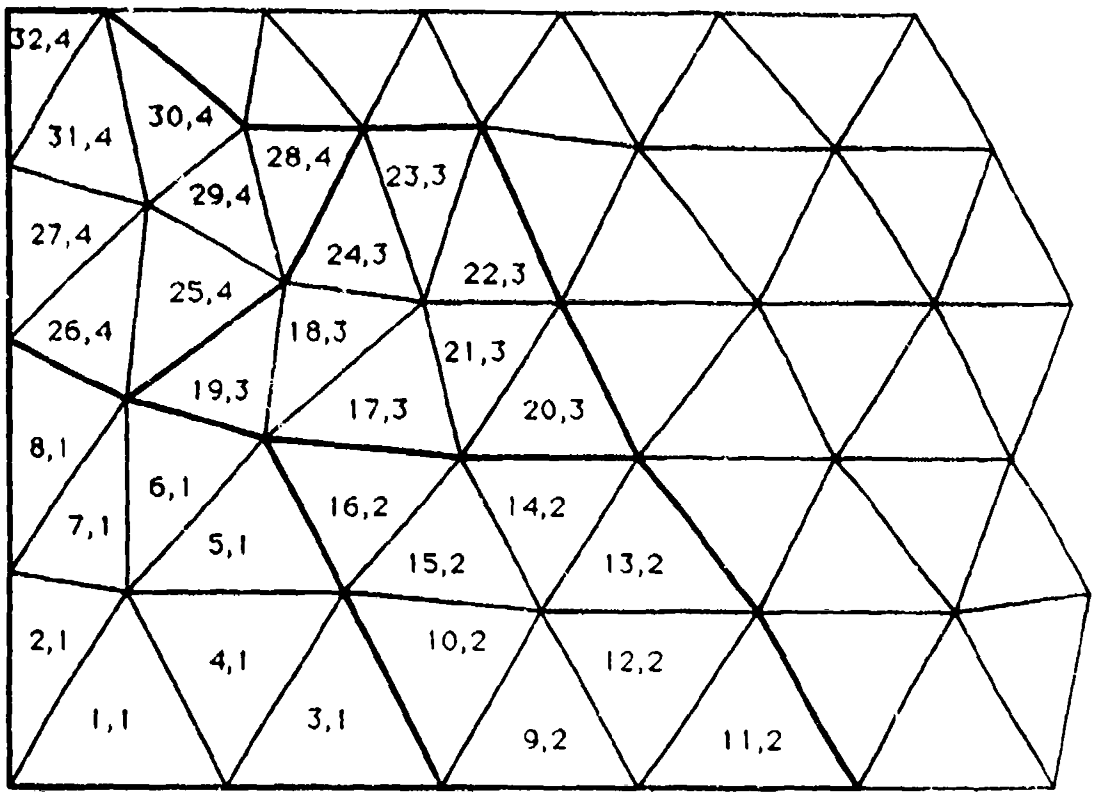


Figure 5.3: Simple Domain Splitting Algorithm
i,j: i: order of creation, j: domain-number

Given the subdomains, there are two possible parallel grid generation strategies (see Figure 5.4):

a) In-Out:
- Grid each subdomain separately;
- Grid pairs of adjacent domains;
- Grid the corners.

One can also work directly with corners after gridding each subdomain. However, it is clear that the achievable parallelism is greater if they are postponed.

b) Out-In:
- Grid the corners;
- Grid pairs of adjacent domains;
- Grid each subdomain separately.

This second approach avoids the need of checking against the borders of the subdomain in the last, and most CPU-intensive step. However, the correct gridding of corners and lines without conflicts is more difficult than in the first method. In the present case, the first approach was implemented.

5.5.1 Generation of Subdomain-Grids

In each subdomain, the advancing-front grid generation technique [Ga.3-7] is used to generate an unstructured grid. Given a background grid that defines the desired spatial distribution of element size and shape, the following steps are required (see Figure 5.5):

F.1 Find the outer faces of the background grid.

F.2 If no active, initial faces are present: generate a first element based on the background grid. This initial element yields a first set of active faces.

F.3 Select the next available active face to be deleted from the front. In order to avoid large elements crossing over regions of small elements, the face forming the smallest new element is selected as the next face to be deleted from the list of faces.

F.4 For the face to be deleted:

F.4.1 Select a 'best point' position for the introduction of a new point `IPNEW`.

F.4.2 Determine whether a point exists in the already generated grid that should be used in lieu of the new point. If there is such a point, set this point to `IPNEW` and continue searching (go to F.4.2).

F.4.3 Determine whether the element formed with the selected point `IPNEW` crosses any given faces. If it does, select a new point as `IPNEW` and try again (go to F.4.3).

F.4.4 Determine whether the element formed with the selected point `IPNEW` crosses any given background faces. If it does, mark the current face as unavailable and select a new face (go to F.8).

F.5 Add the new element, point, and faces to their respective lists.

F.6 Find the generation parameters for the new faces from the background grid.

F.7 Delete the known faces from the list of faces.

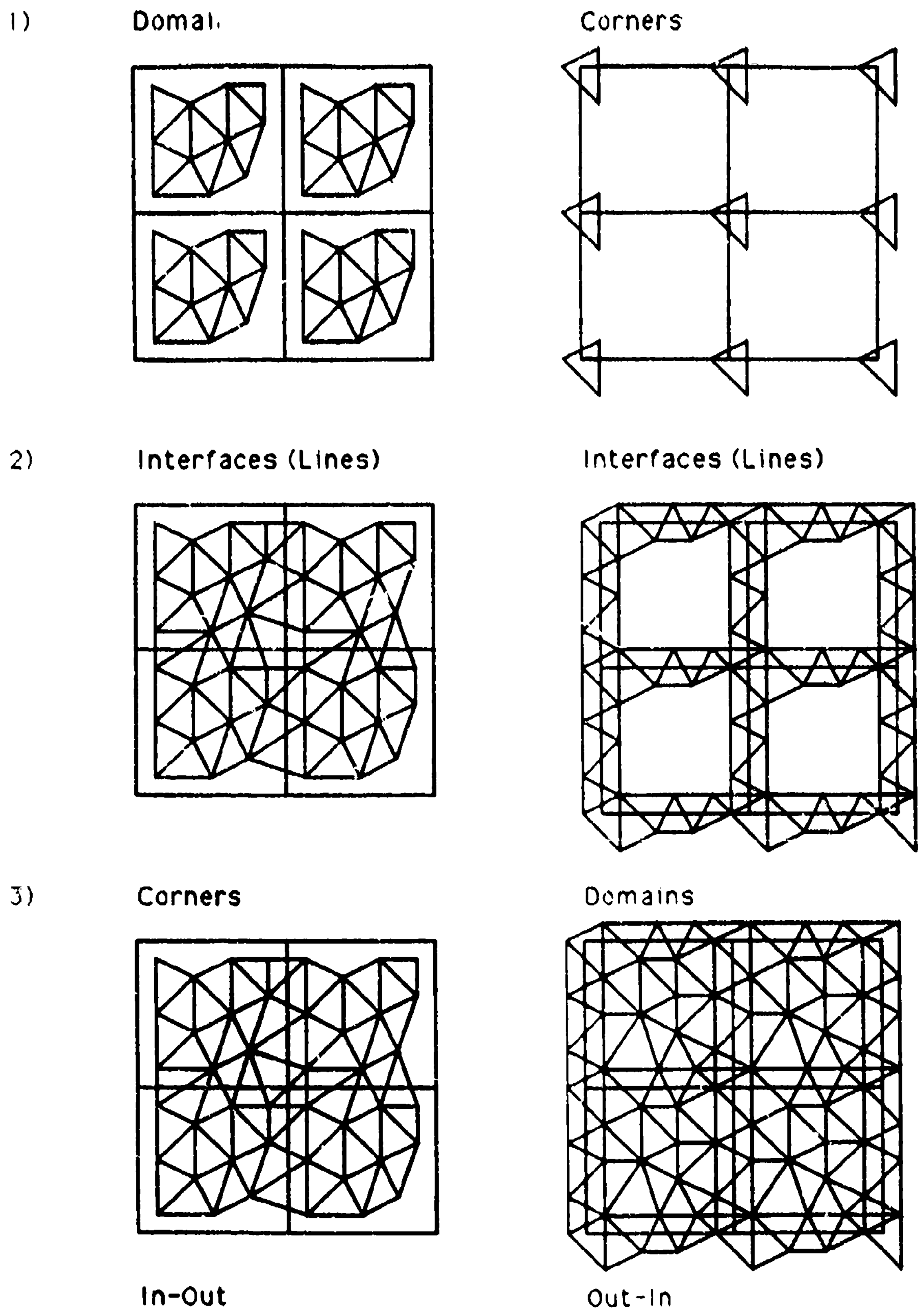


Figure 5.4: Parallel Grid Generation Strategies

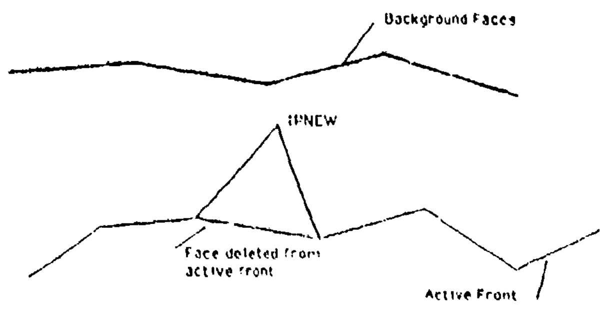


Figure 5.5: Advancing Front Grid Generation in Each Subdomain

F.8 If there are any active, available faces left in the front, go to F.3.

Compared to the usual advancing front method, the only modifications required are steps F.1, F.2 and F.4.4.

5.5.2 Data Management

5.5.2.1 Data Structures for the Host

In order to simplify the complexity of data handling as much as possible, the following rules were followed:
- A point belongs uniquely to one (and only one) subdomain;
- An element belongs to the lowest domain-number of its nodes;
- A face may belong to more than one domain.

In the global arrays that store the mesh as it is assembled, the subdomains have allotted slots. The storage scheme used is shown diagrammatically for the coordinates in Figure 5.6.

Figure 5.6: Storage of Coordinate Information in the Master-Node

5.5.2.2 Data Structures for the Node

No special data structures are required for the node. In fact, the node-code is the same as the one used for a single-processor machine.

5.5.3 Information Flow

The information transfer required for the parallel grid generation algorithm described above is as follows:

From the Host to the Node:
- Assemble the required information for the domains involved:
  - Background Grids
  - Active Faces
  - Points of these Faces
- Renumber to obtain local arrays, remembering the renumbering order;
- Send the assembled information to the node.

From the Node to the Host:
- Obtain the remaining information from the node:
  - Active Faces
  - New Elements
  - New Points
- Renumber and store in the global arrays.

5.5.4 Contingency Tests

When generating the inter-domain regions, one can not generate at the same time the interfaces of all neighbors for a certain subdomain. Therefore, a contingency list was implemented that avoids these conflicts. It operates on the premise that during each task hand-out pass over the interdomain-boundaries or corners, a subdomain may be touched once only. During each task hand-out pass, the following operations are performed:

```
C.1 Initialize a subdomain-array;
C.2 Loop over the interdomain-boundaries (or corners):
        If none of the subdomains involved has already been used:
            - Mark the subdomains involved;
            - Renumber and send to the next available processor the
              information necessary;
        Endif
```

5.6 Parallel Smoothing and Explicit Flow Solvers

Practical implementations of either advancing front or Voronoi grid generators indicate that in certain regions of the mesh abrupt variations in element shape or size may be present. These variations appear even when trying to generate perfectly uniform grids. The usual way to circumvent this problem is to improve the uniformity of the mesh by smoothing. The most commonly used smoother is the so-called Laplacian smoother. The sides of the triangulation are assumed to represent springs. These springs are then relaxed in time using explicit time stepping, until an equilibrium of spring-forces has been established. The flow of information and most of the loops in such a smoother are equivalent to an explicit time-marching scheme. Therefore, the techniques used for the smoother and the flow solver are identical. The implementation of a spring analogy smoother or an explicit flow solver on a parallel machine requires the following steps:

```
S.1 Subdivide the given mesh into subdomains;
S.2 For each smoothing pass or timestep:
        - Smooth each subdomain separately;
        - Exchange boundary information;
S.3 Assemble the result.
```

Thus, for the smoother and the flow solver an equal-level hierarchy is employed. This seems appropriate, as

a) no contingency tests (other than the synchronization at the end of one pass over the elements) are required, and
b) the amount of CPU spent is roughly the same in each processor.

5.6.1 Smoothing of Subdomain-Grids

In each subdomain, the standard Laplacian smoother is employed. Each side of the triangulation is supposed to represent a string. Thus, the force acting on each point is given by:

$$\mathbf{f}_i = c\sum_{j=1}^{ns_i}(\mathbf{x}_j - \mathbf{x}_i) \quad , \qquad (5.2)$$

where c denotes the spring constant, $\mathbf{x}_0$ the coordinates of the point, and the sum extends over all the points surrounding the point. The time-advancement for the coordinates is accomplished as follows:

$$\Delta\mathbf{x}_i = \Delta t\frac{1}{ns_i}\mathbf{f}_i \quad . \qquad (5.3)$$

At the outer boundary of the subdomain, $\Delta\mathbf{x} = 0$. Usually, 3-4 timesteps or passes over the mesh yield an acceptable mesh.

5.6.2 Data Structures

In order to be able to also move the boundary points or change the unknowns at subdomain interfaces, an exchange of information between processors has to be allowed. Two possible choices are possible (see Figure 5.7):

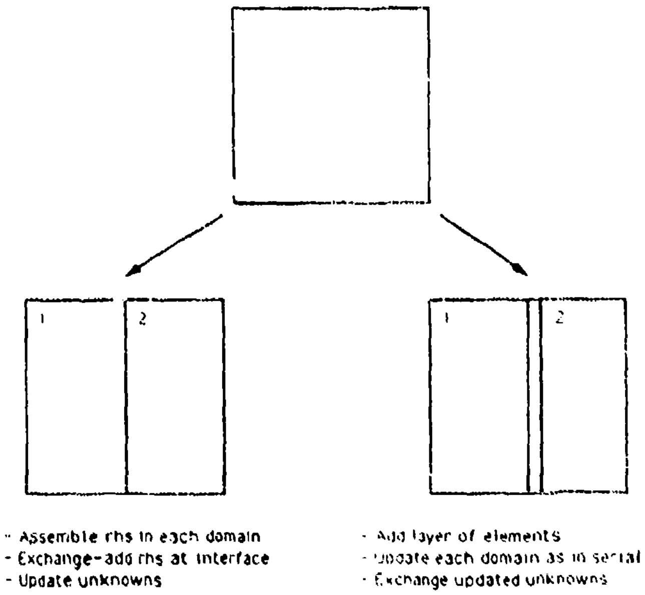

1
2
1
2
- Assemble rhs in each domain
- Exchange-add rhs at interface
- Update unknowns
- Add layer of elements
- Update each domain as in serial
- Exchange updated unknowns

Figure 5.7: Updating Strategies

a) Right-Hand Side Information:

- Assemble all force contributions (right-hand sides) in each processor;
- Exchange force contributions between processors, adding;
- Update the coordinate vectors of the interfaces.

The advantage of this approach is that the total amount of information (elements, points) required for the parallel version is the same as for the serial version. However, the original serial code has to be modified extensively.

b) Coordinate Information:

- Add a further layer of elements to each subdomain;
- Update each subdomain as for the serial, 1-domain case;
- Exchange the updated coordinate information between processors.

In this approach, the total amount of information (elements, points) required for the parallel version is larger than that required for the serial version. However, the extra amount of information is not large (1 layer of elements). On the other hand, the code employed in each subdomain is exactly the same as for the serial case. It was felt that this advantage outweighs all possible disadvantages.

This second method requires the addition of layers of elements at the boundaries of the subdomains. The following algorithm accomplishes this task:

Assume given:

- The elements that surround each point;
- The domain-nr. each element belongs to.

Then:

```
L.1 Initialize pointer-lists for elements, points and
    receive-lists;
L.2 For each point IPOIN:
    Get the smallest domain-nr. IDMIN of the el-
    ements that surround it; store this number in
    LPMIN(IPOIN);
        For each element that surrounds this point:
        If the domain-nr. of this element is larger
        than IDMIN :
                - Add this element to domain IDMIN
L.3 For the points of each subdomain IDOMN:
    If LPMIN(IPOIN).NE.IDOMN :
        add this information to the receive-list for
        this subdomain;
    Endif
L.4 Order the receive-list of each subdomain accord-
    ing to subdomains;
L.5 Given the receive-lists, build the send-list for
    each subdomain.
```

5.6.3 Information Flow

The information transfer required for the parallel smoothing or flow advancement algorithm described above is as follows:

From the Host to the Node:

- Assemble the required information for the do-

mains involved:
- Grid
- Send/Receive Lists

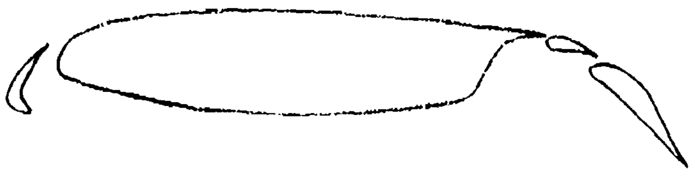
- Send the assembled information to the node.

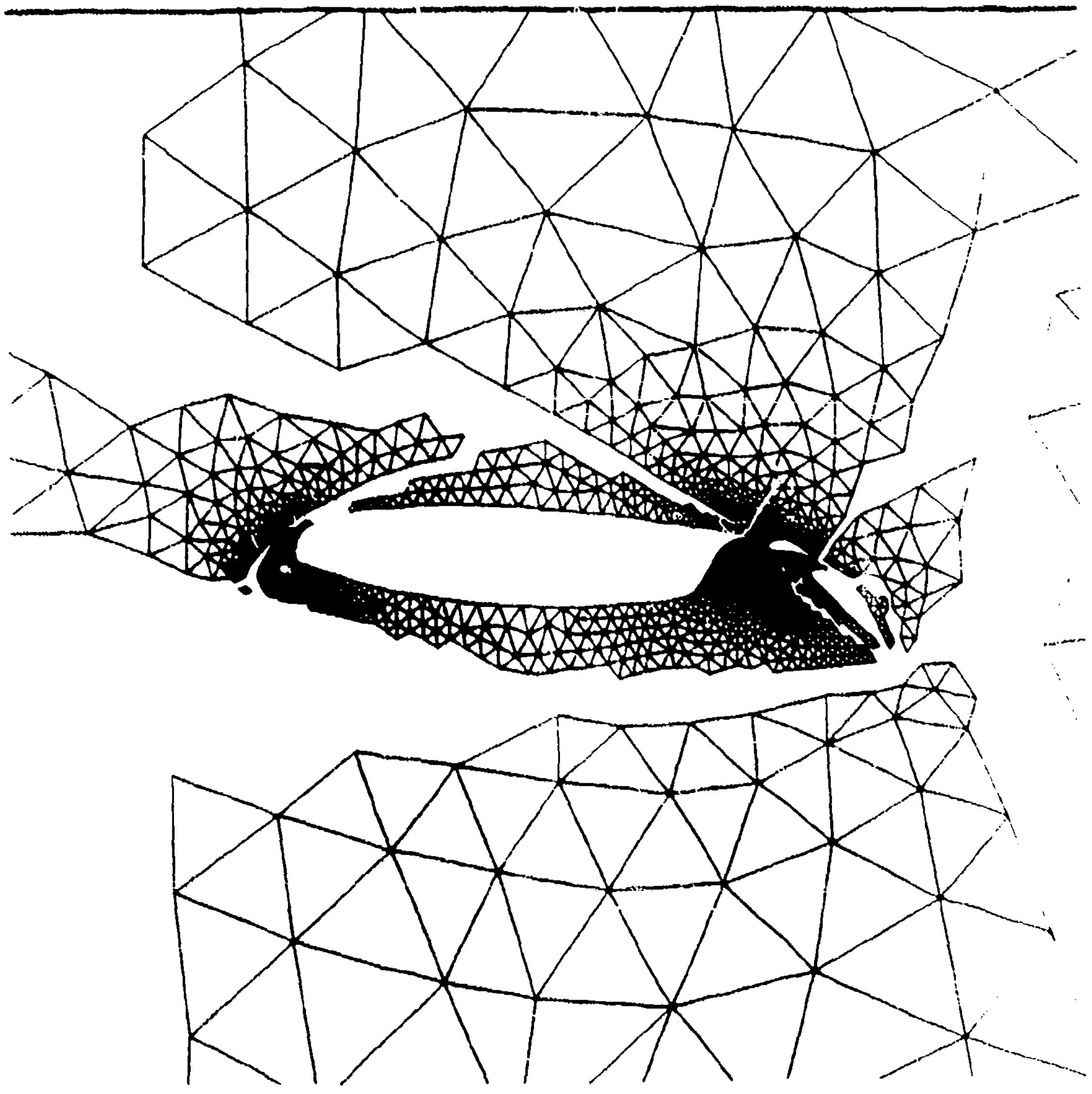
From Node to Node:

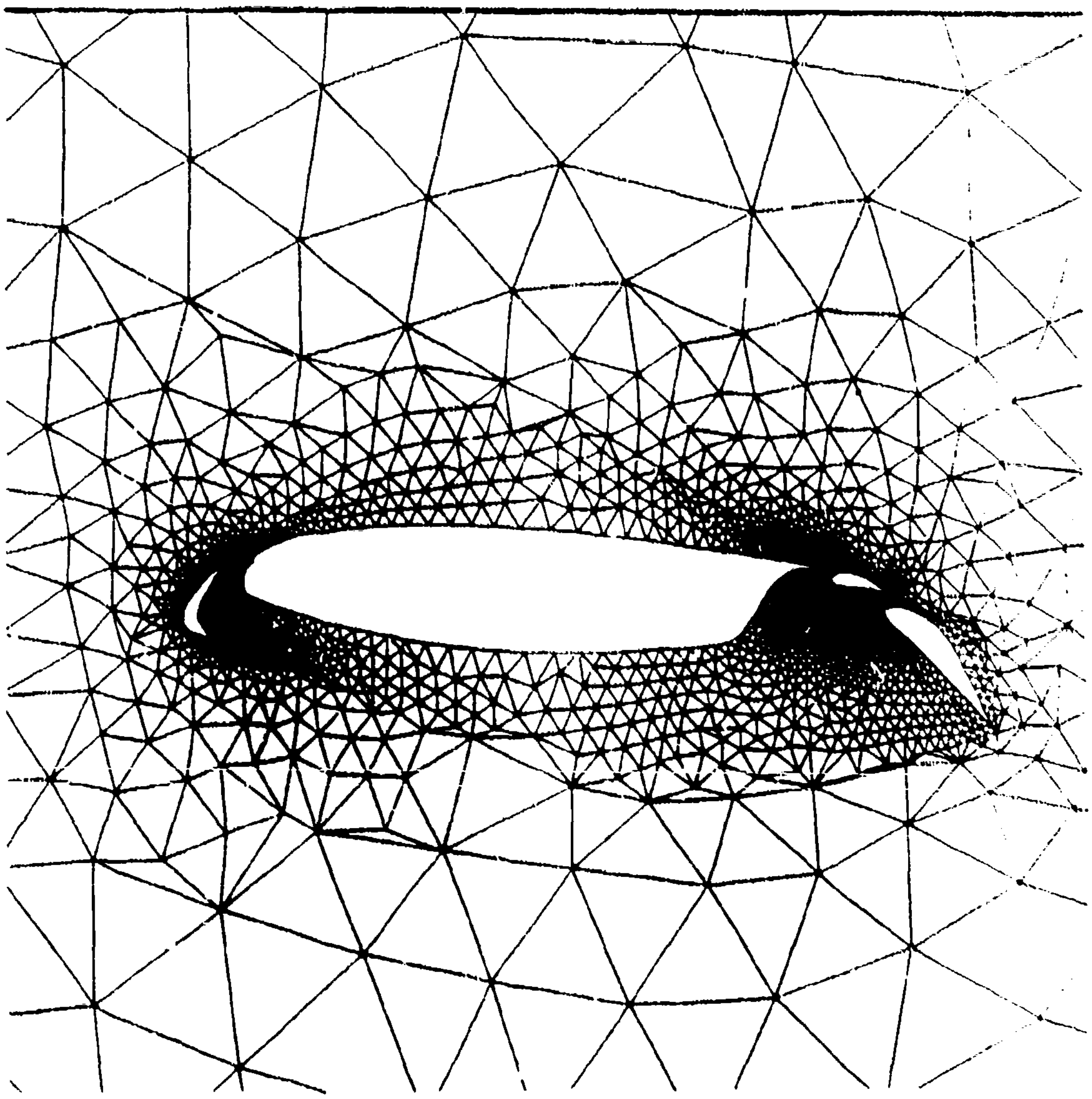
- Send the updated coordinates of all nodes stored in the send-list;
- Receive the updated coordinates of all nodes stored in the receive-list;
- Overwrite the coordinates for these received points.

From the Node to the Host:

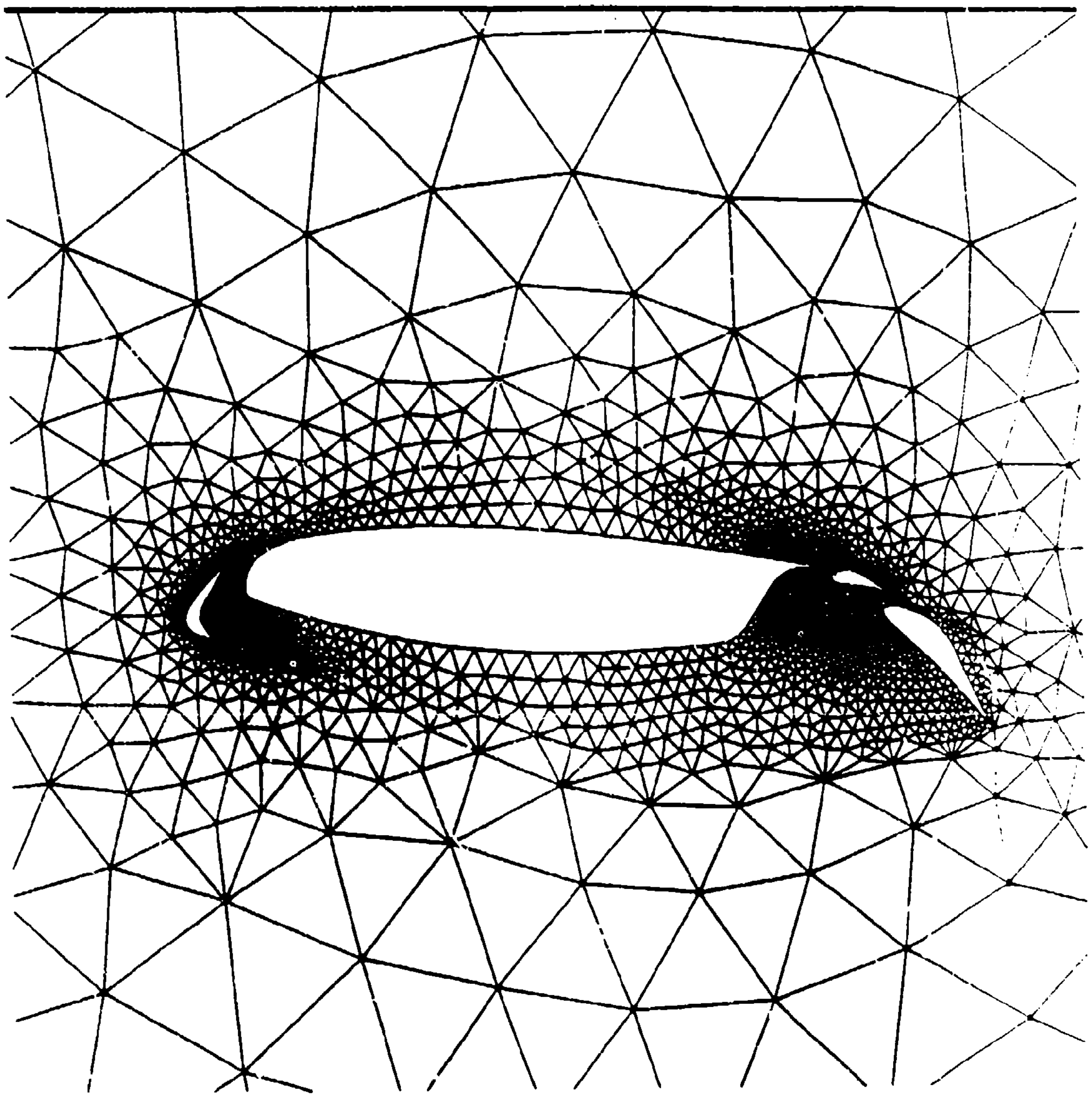
- Remember and store in the global arrays.


Multi Element Airfoil: Domain Definition


Multi Element Airfoil: Gridded Subdomains



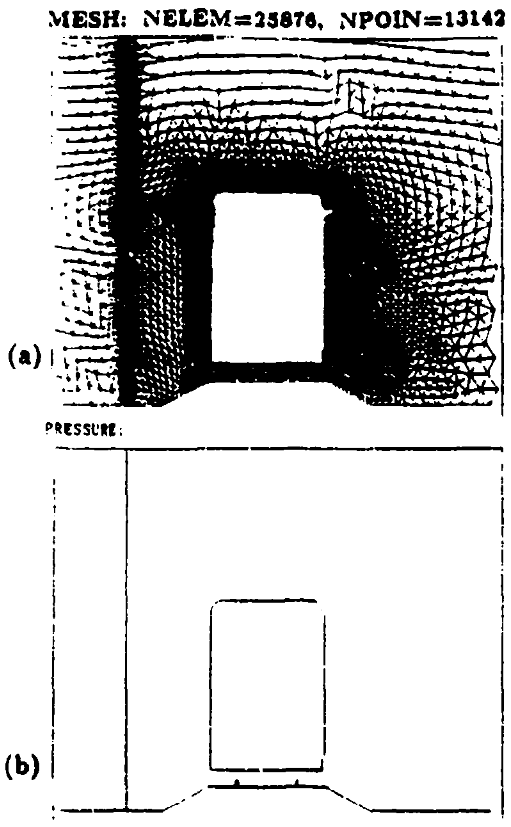
Multi Element Airfoil: Final Assembled Mesh



Multi Element Airfoil: Smoothed Mesh

Examples 1 **Shock Interaction with an Elevated Box**

- Grid Generator: FRGEN2D
- Hydro-Code: FEFLO27 (2-D, Euler, H-Refinement)
- Plotting: FEPLOT2D
- Details: [Ae.6, Ae.7]



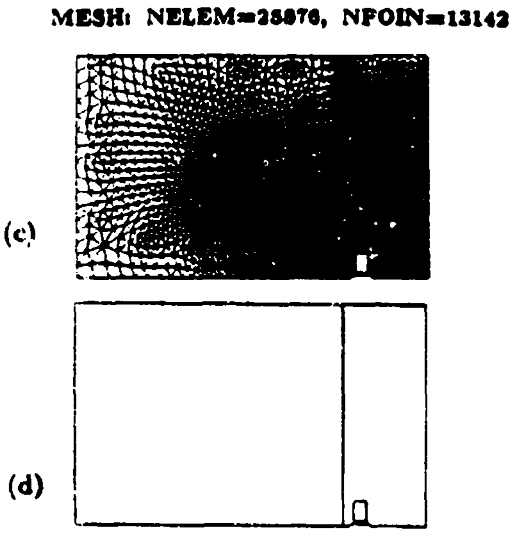


Fig 1. Expanded views of (a) Mesh and (b) Pressure Contours Around the Box; (c) Mesh and (d) Pressure Contours for the Complete Computational Domain. t=0.0.

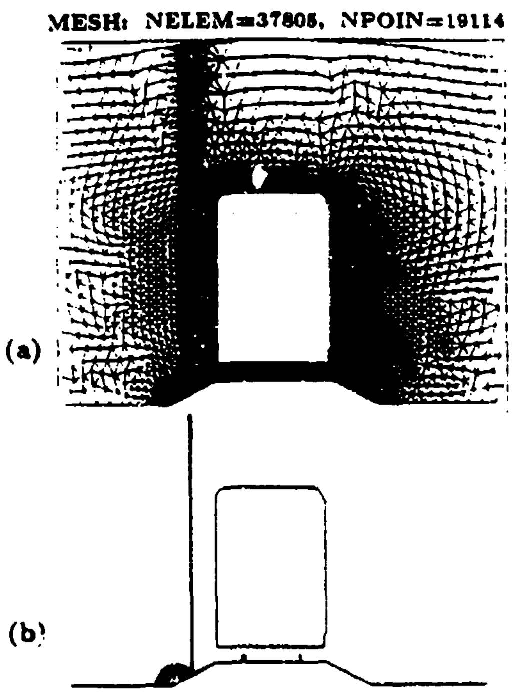


Fig 2. Expanded Views of (a) Mesh and (b) Pressure Contours Around the Box, t=0.4.

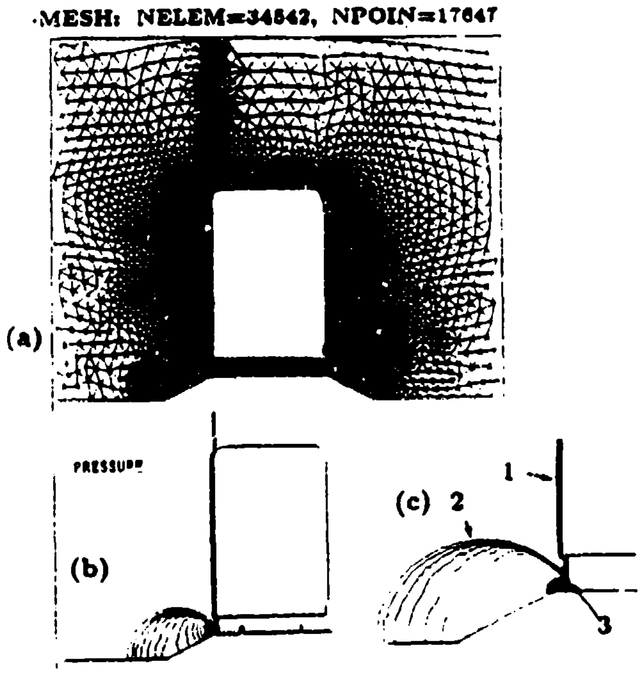
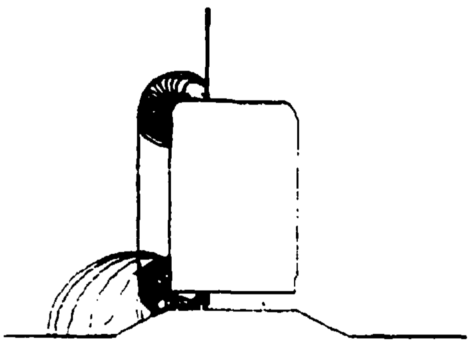


Fig 3. Expanded Views of (a) Mesh, (b) Pressure Contours Around the Box, and (c) Pressure Contours Under the Box. t=0.6

(a)



(b)

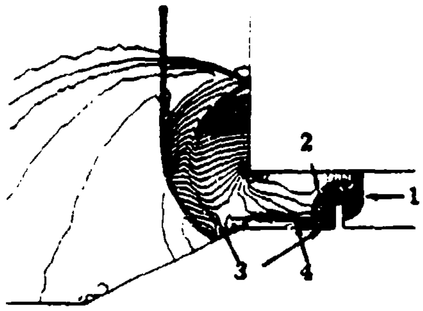
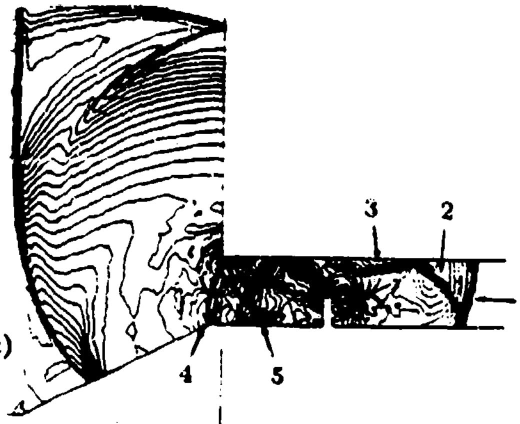


Fig 4. Pressure Contours (a) Around and (b) Under the Box, t=0.8

(c)



(d)

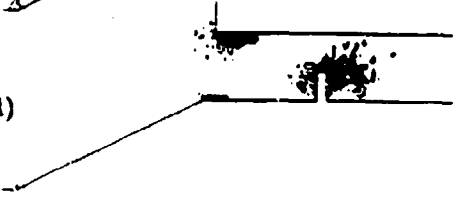
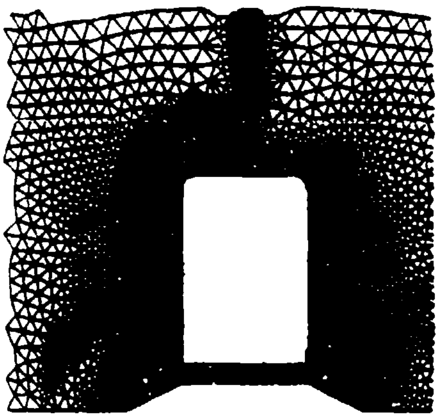


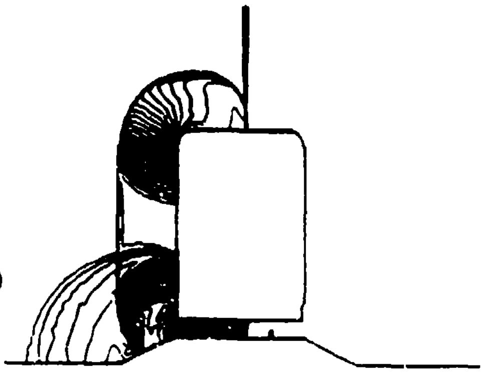
Fig 5. Expanded Views of (a) Mesh and (b) Pressure Contours Around the Box; (c) Pressure and (d) Vorticity Contours Under the Box, t=1.0.

MESH: NELEM=63544, NPOIN=31823

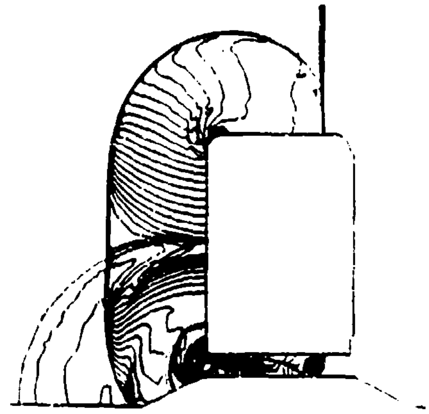
(a)



(b)



(a)



(b)

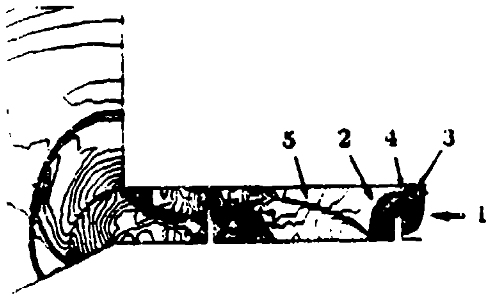


Fig 6. Pressure Contours (a) Around and (b) Under the Box, t=1.2.

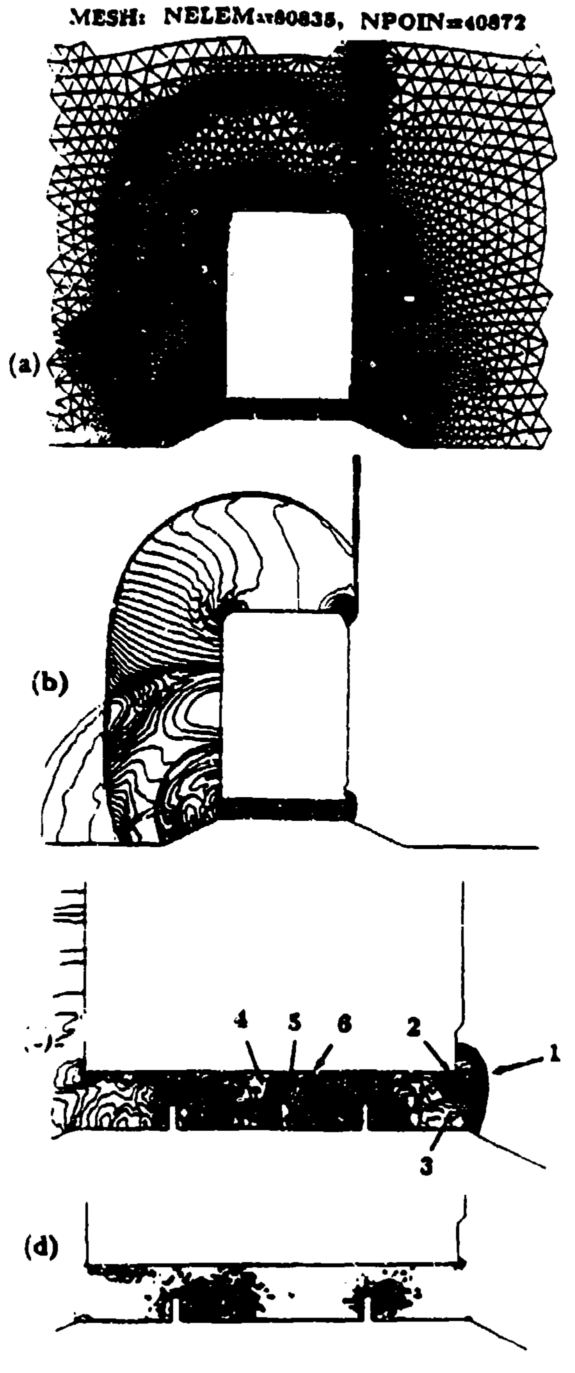


Fig 7. Expanded Views of (a) Mesh and (b) Pressure Contours Around the Box; (c) Pressure and (d) Vorticity Contours Under the Box, t=1.4.

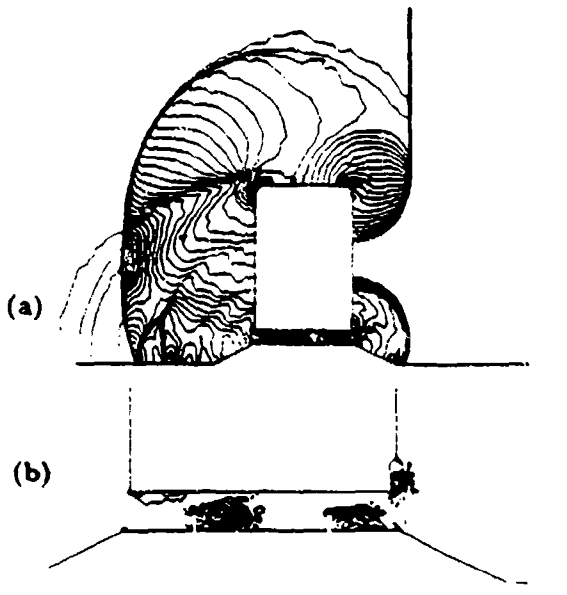
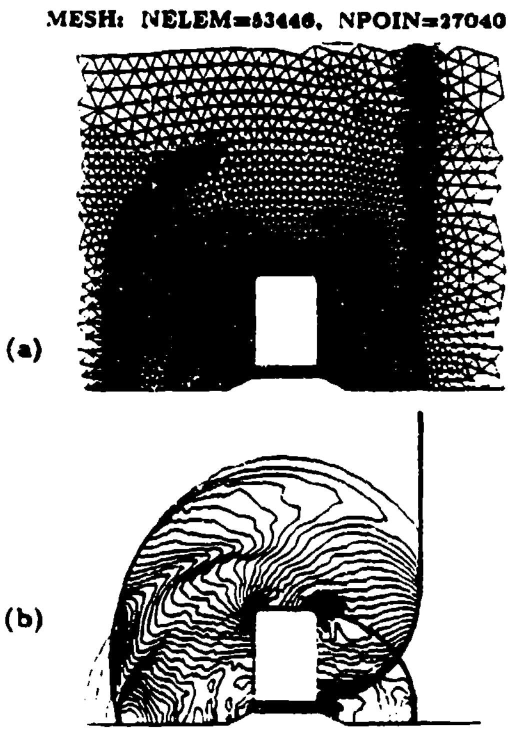


Fig 8. (a) Pressure Contours Around the Box; (b) Vorticity Contours Under the Box. t=1.8.



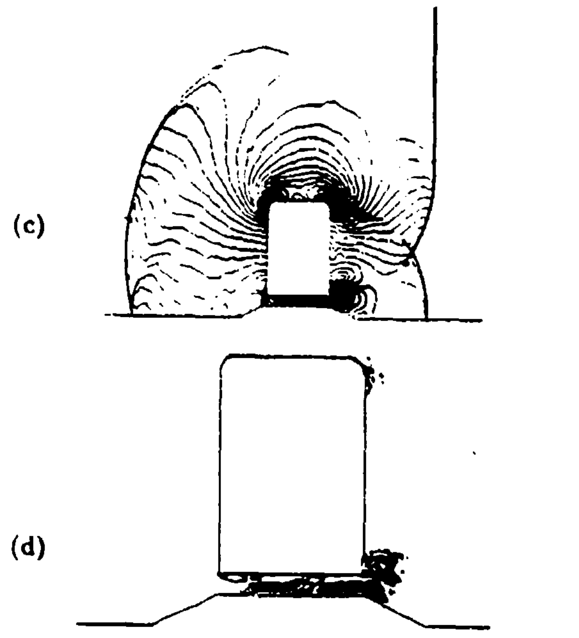


Fig 9. Expanded Views of (a) Mesh, (b) Pressure, (c) Mach Number and (d) Vorticity Contours Around the Box, t=2.6.

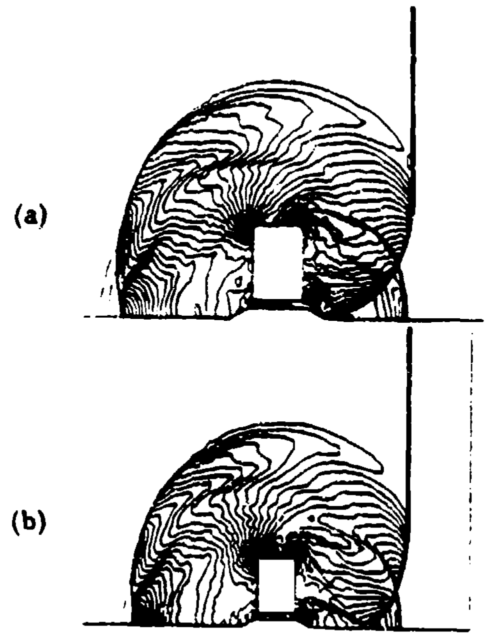


Fig 10. Expanded Views of Pressure Contours Around the Box: (a) t=3.2; (b) t=3.6.

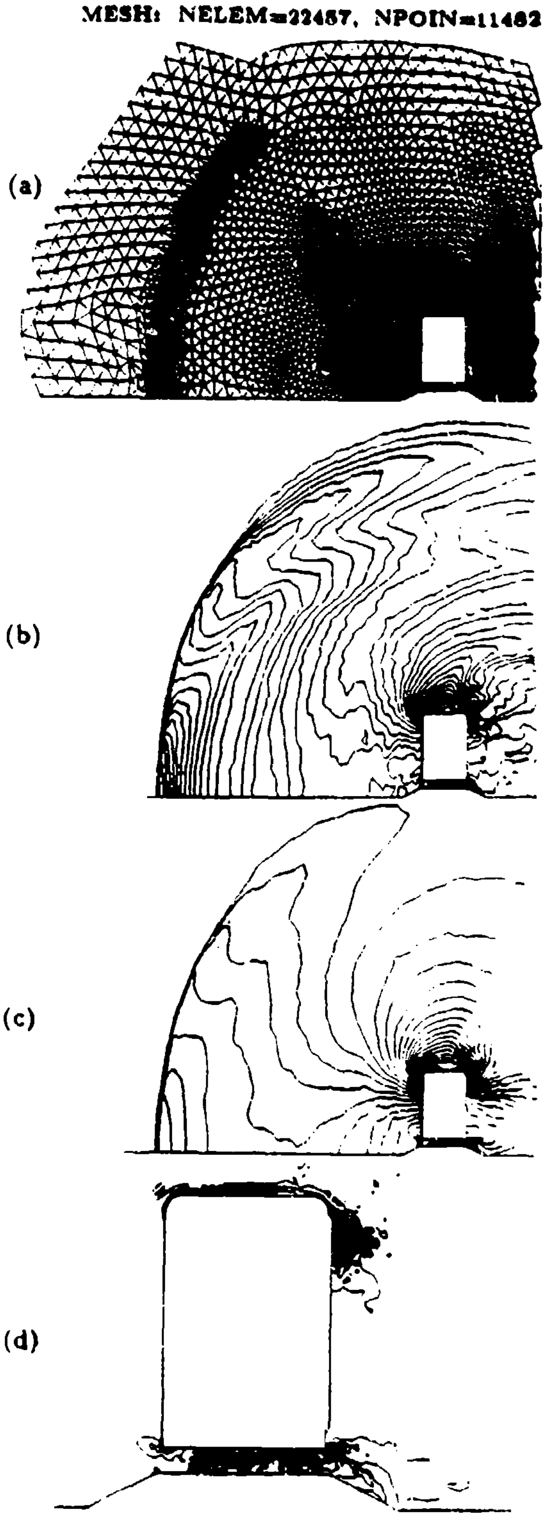


Fig 11. Expanded Views of (a) Mesh, (b) Pressure, (c) Mach Number and (d) Vorticity Contours Around the Box, t=6.0.

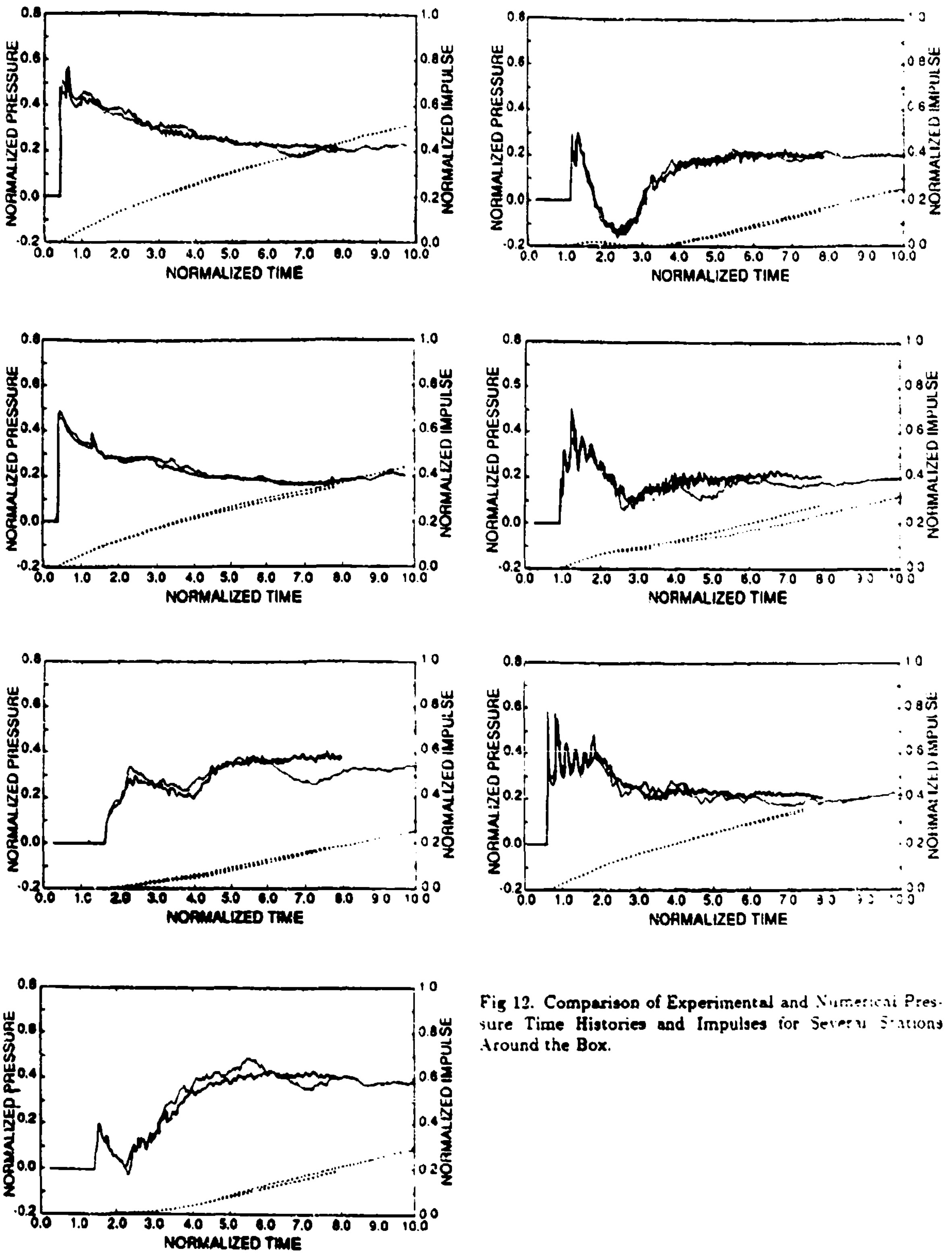
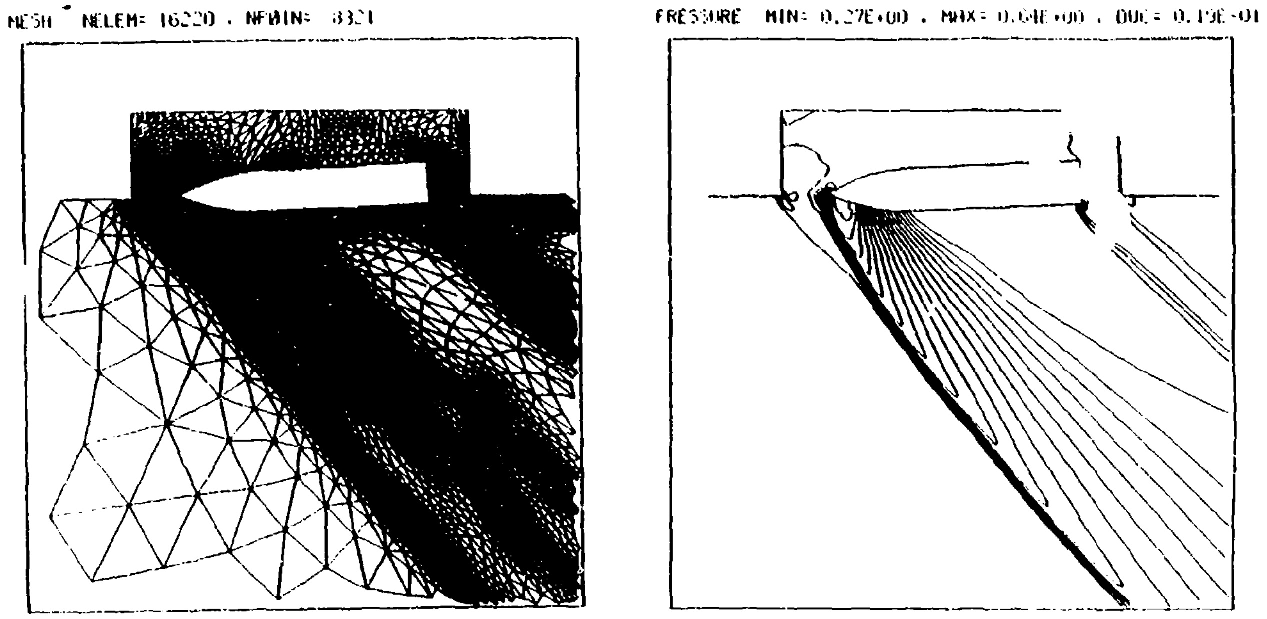


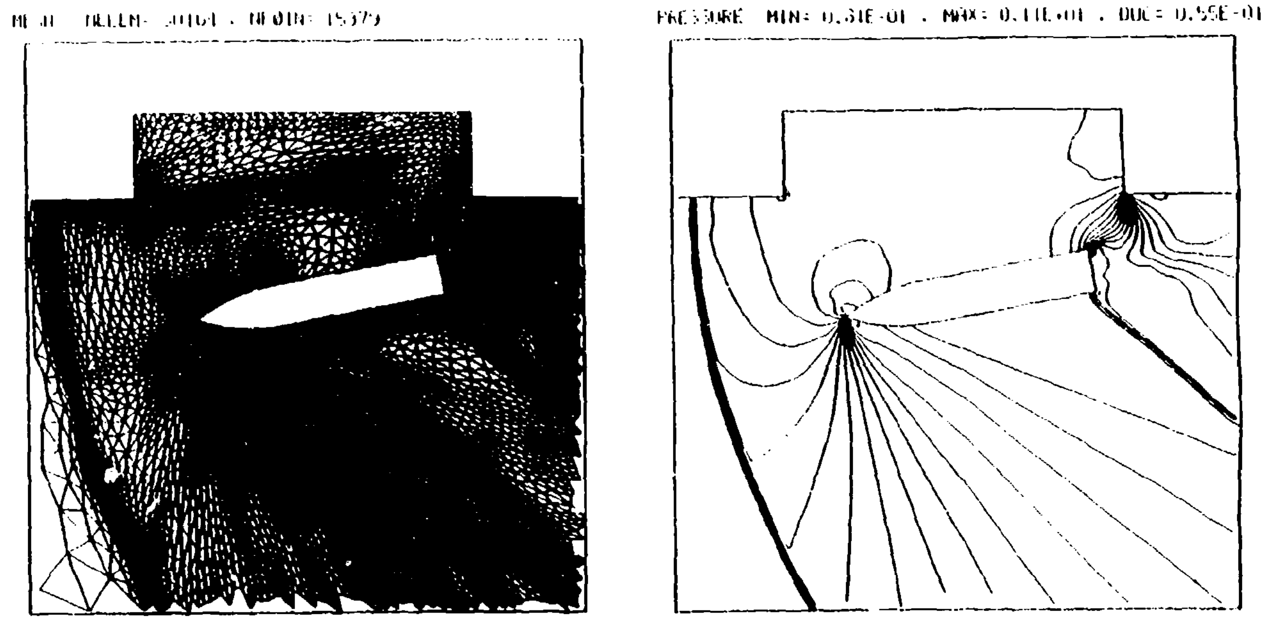
Fig 12. Comparison of Experimental and Numerical Pressure Time Histories and Impulses for Several Stations Around the Box.

Examples 2 **Store Release into Supersonic Free Stream**

- Grid Generator: FRGEN2D
- Hydro-Code: FEFLO44 (2-D, ALE-Euler, Remeshing)
- Plotting: FEPLOT2D
- Details: [Ar.2, Ar.3]


MESH NELEM= 16220 . NPOIN= 8321
PRESSURE MIN= 0.27E+00 . MAX= 0.64E+00 . DUC= 0.19E-01

Store Separation Problem. T=

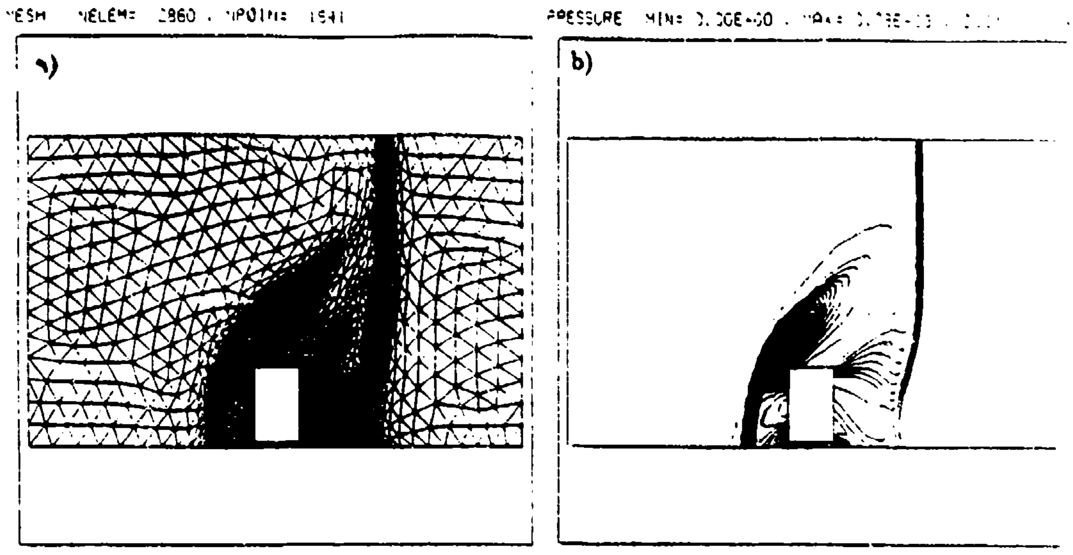


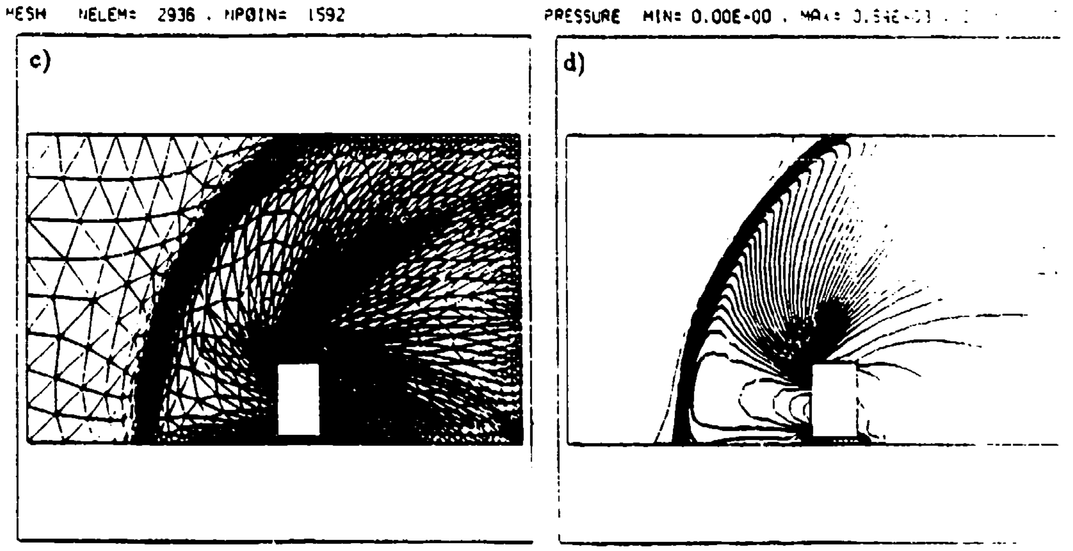
Store Separation Problem. T=

Examples 3

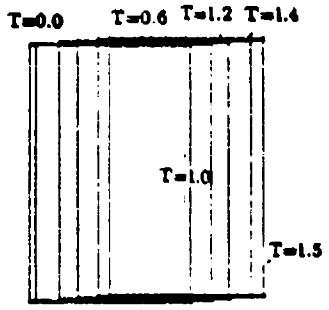
Shock Interaction with a Box

- Grid Generator: FRGEN2D
- Hydro-Code: FEFLO44 (2-D, ALE-Euler, Remeshing)
- Plotting: FEPLOT2D
- Kinematic Condition: Free Flight After Onset of Lift
- Details: [Ar.2, Ar.3]





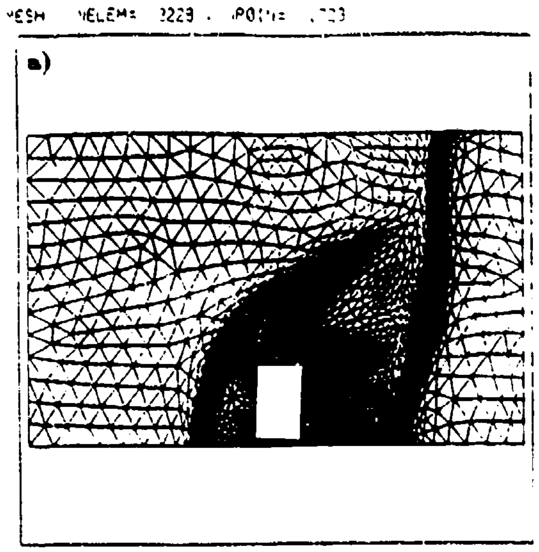
e) Position of the Box in Time

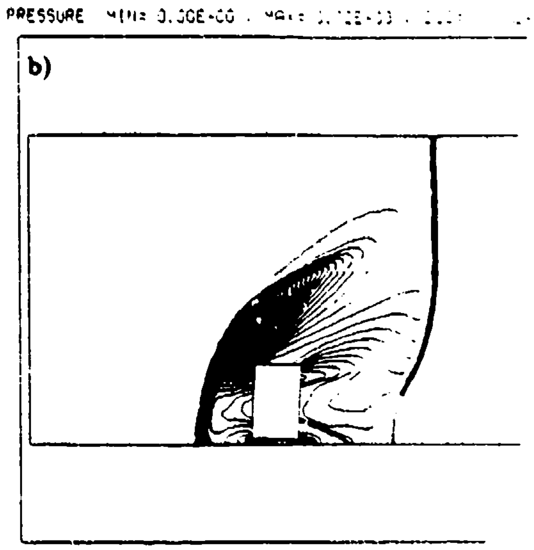


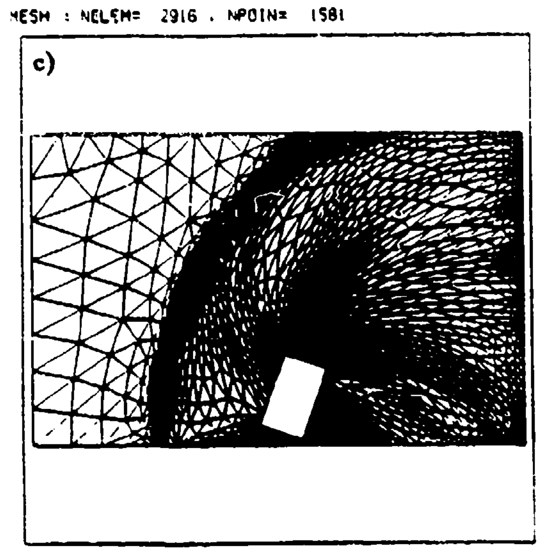
Shock-Box Interaction. Free B.

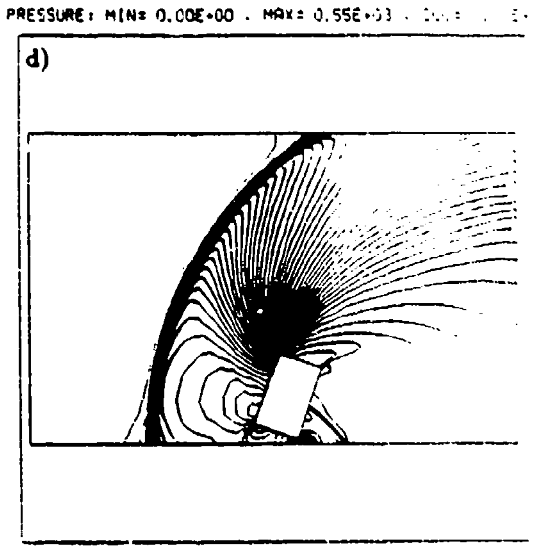
Examples 4 **Shock Interaction with a Box**

- Grid Generator: FRGEN2D
- Hydro-Code: FEFLO44 (2-D, ALE-Euler, Remeshing)
- Plotting: FEPLOT2D
- Kinematic Condition: Constrained Until Onset of Lift
- Details: [Ar.2, Ar.3]

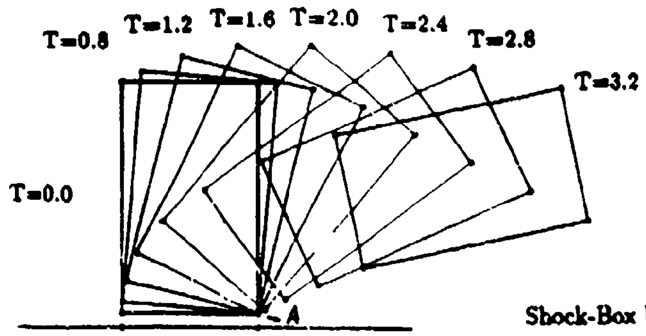








e) Position of the Box in Time

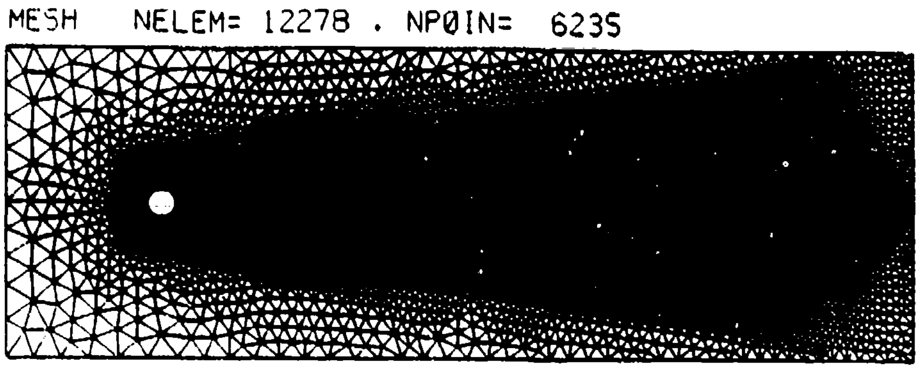


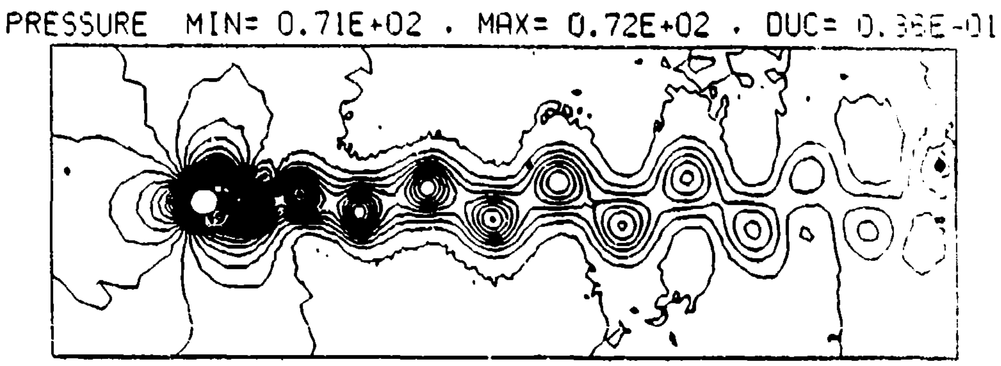
Shock-Box Interaction: Constrained Box

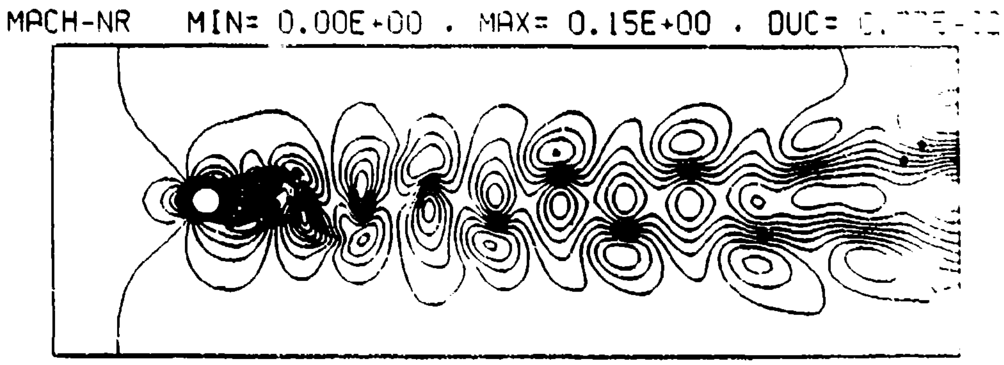
Examples 5

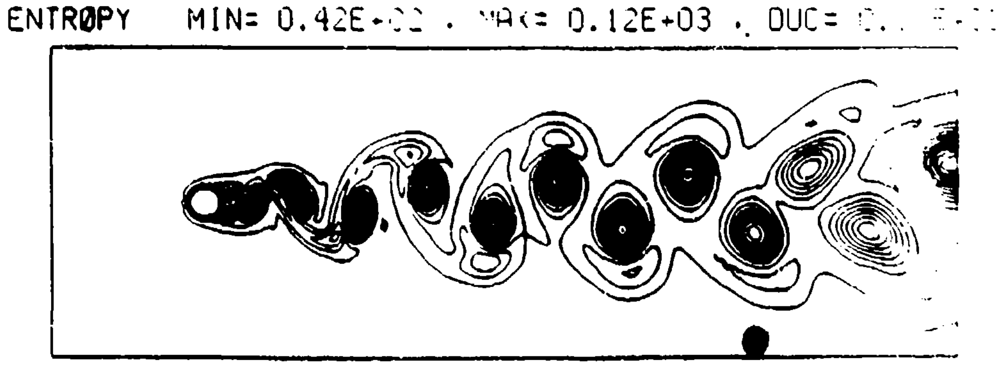
von Karman Vortex Street

- Grid Generator: FRGEN2D
- Hydro-Code: FEBIC2D (2-D, Semi-Implicit)
- Plotting: FEPLOT2D
- Parameters: Ma=0.1, Re=100
- Details: [Hawaii 1989 AIAA CFD Conf.]

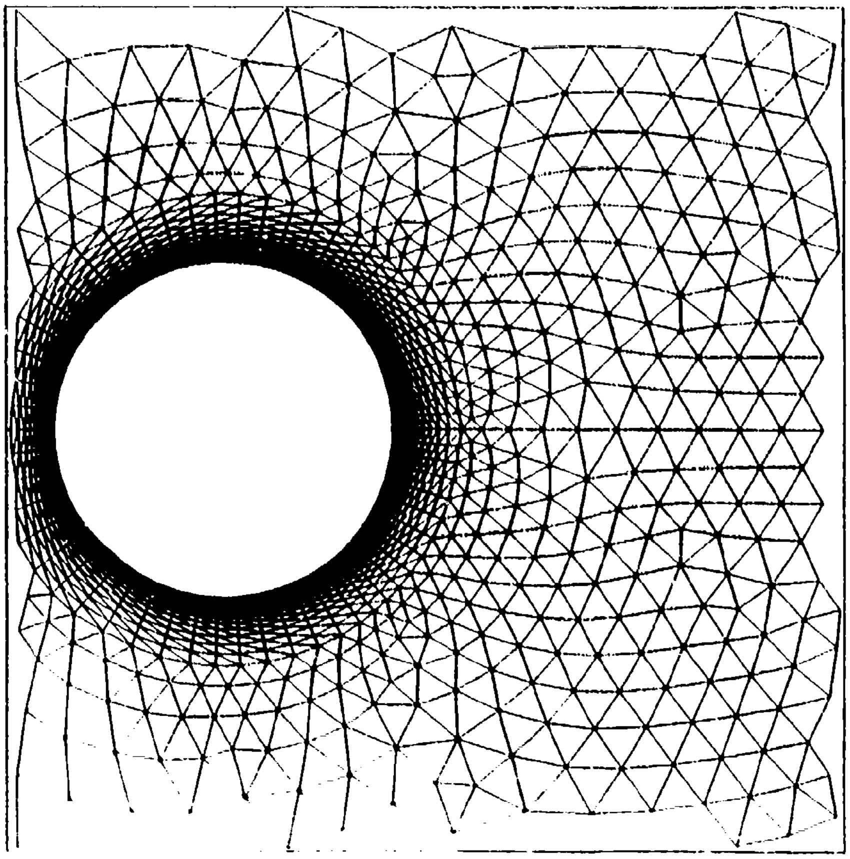




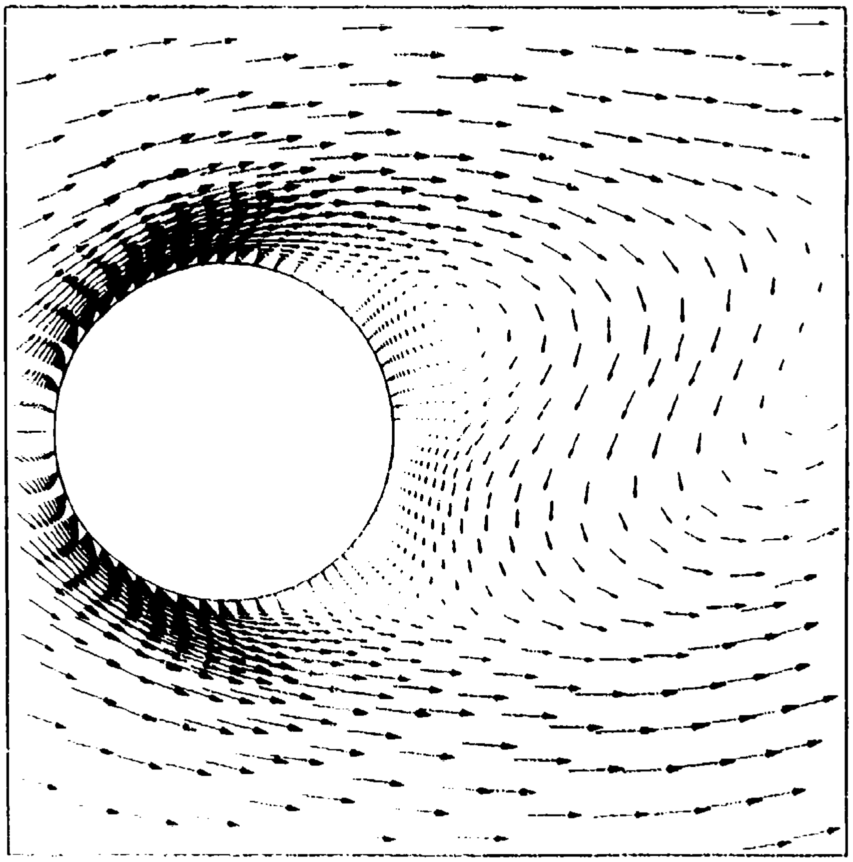




MESH NELEM= 12278 , NPØIN= 6235

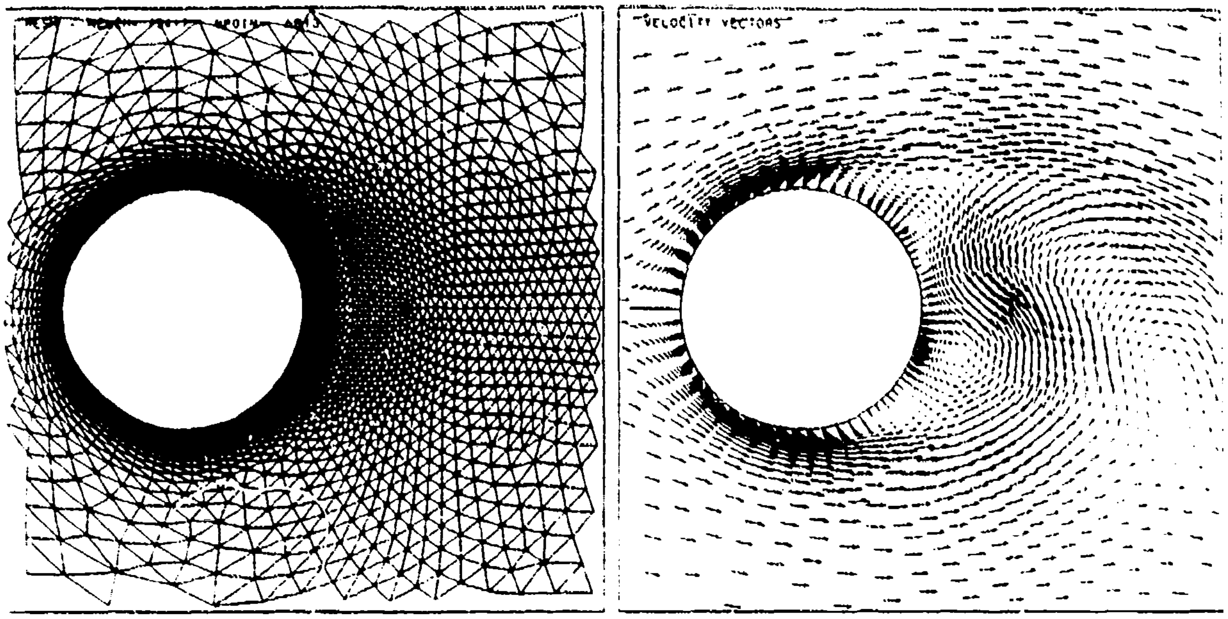


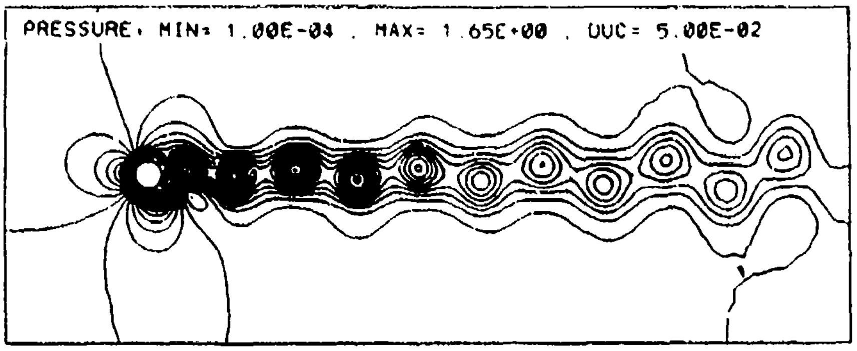
VELØCITY VECTØRS

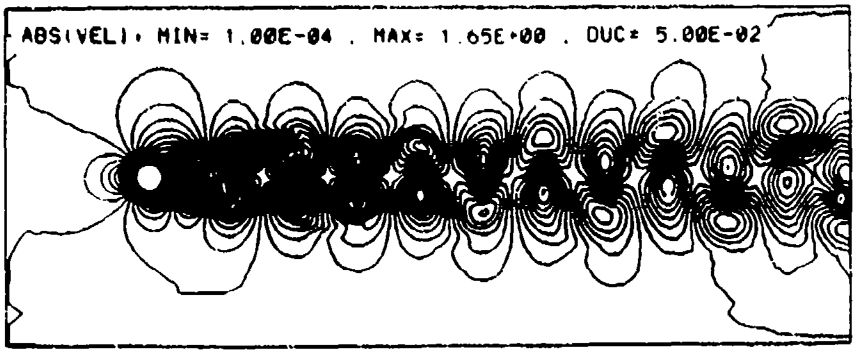


Examples 6 **von Karman Vortex Street**

- Grid Generator: FRGEN2D
- Hydro-Code: FEFLOIC26 (2-D, Incompressible, H-ref.)
- Plotting: FEPLOT2D
- Parameters: Re=1000
- Details: [Sl.2]



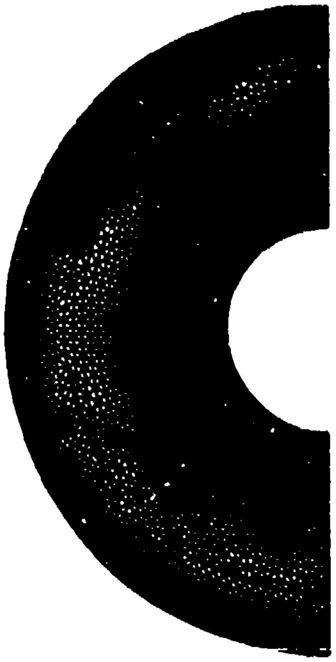


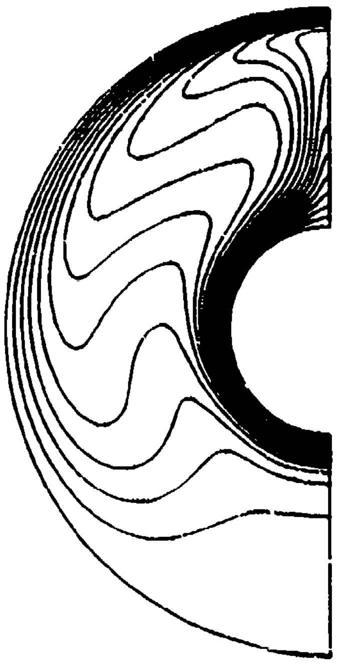


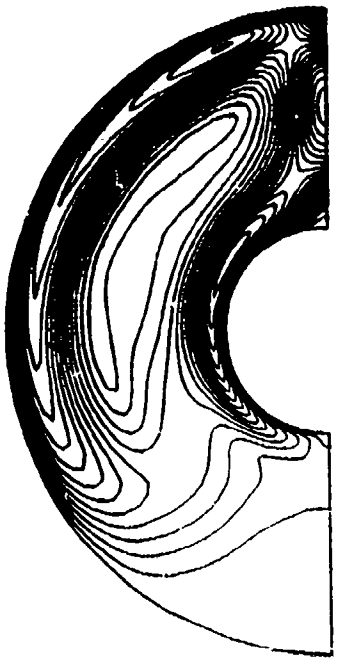
von Karman Vortex Street at Re=1,000 (FEFLOIC25)

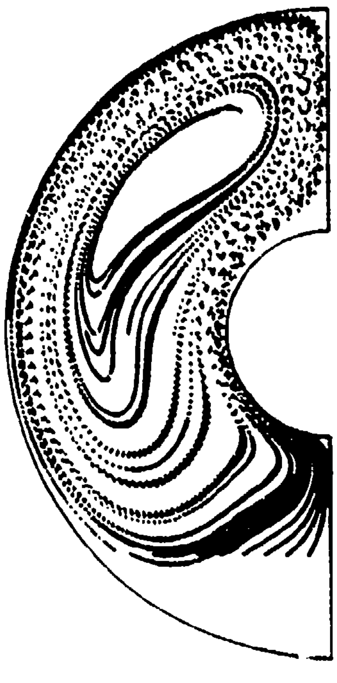
Convection Between Concentric Cylinders

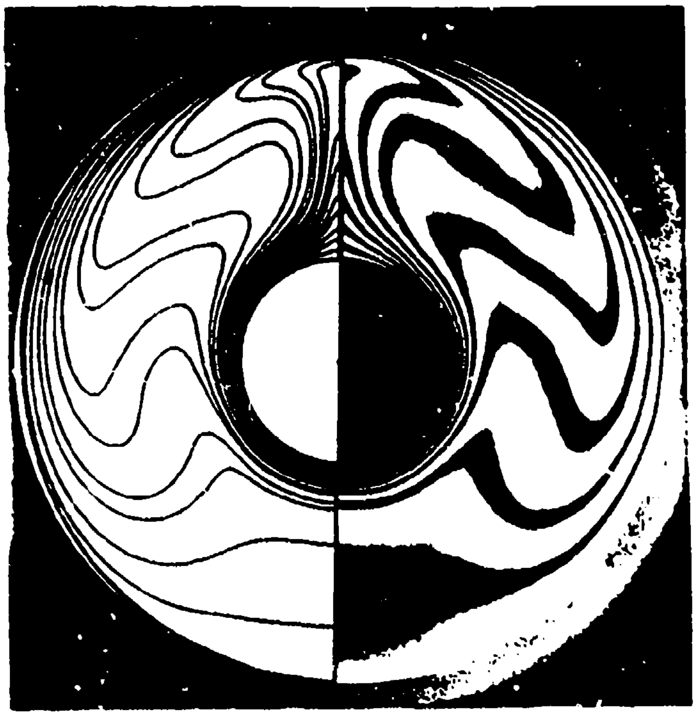
- Grid Generator: FRGEN2D
- Hydro-Code: FEFLOIC26 (2-D, Incompressible, H-Ref.)
- Plotting: FEPLOT2D
- Parameters: $d_o/d_i = 3.14$, $Gr = 122,000$, $Re = 1.0$, $Pr = 0.71$


MESH


TEMPER. DUC= 5.00E-02

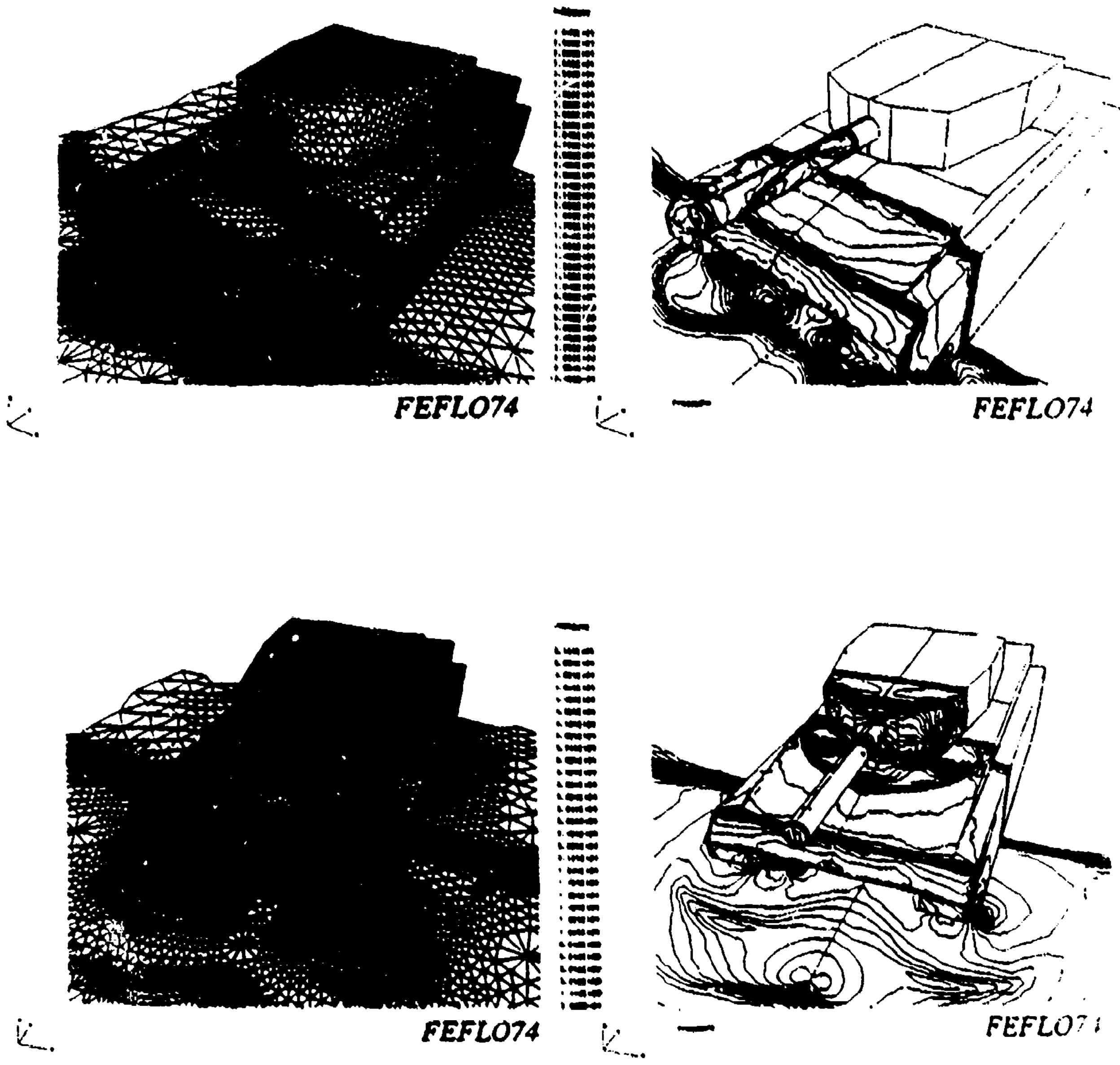

ABS(VEL). DUC= 1.00E+01


PARTICLE TRACES


Convection Between Concentric Cylinders
($Gr = 122,000$, $Re = 1.0$, $Pr = 0.71$)

Examples 8 **Shock Interaction with an Idealized Leopard-2 Tank**

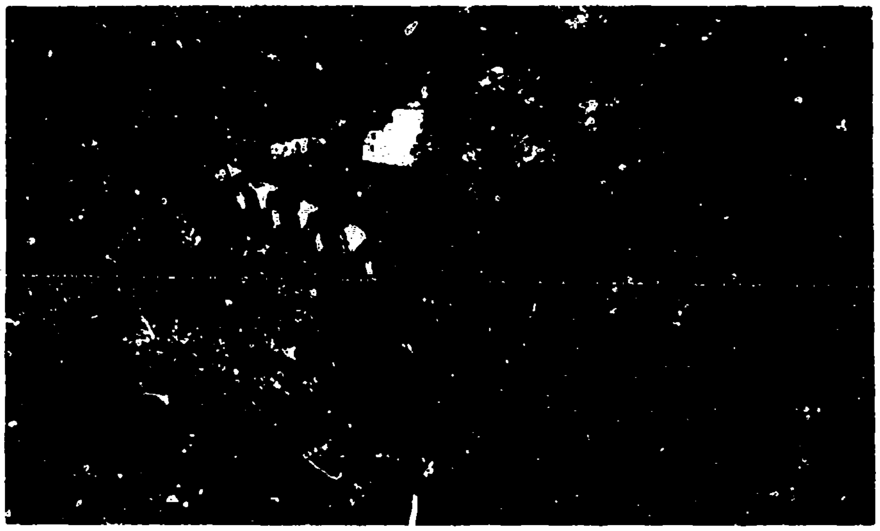
- Grid Generator: FRGEN3D
- Hydro-Code: FEFLO74 (3-D, ALE-Euler, H-Ref.)
- Plotting: FEPLOT3D, FEPOST3D
- Details: [Ae.7, Ae.9]



Shock-Tank Interaction (FEFLO74)

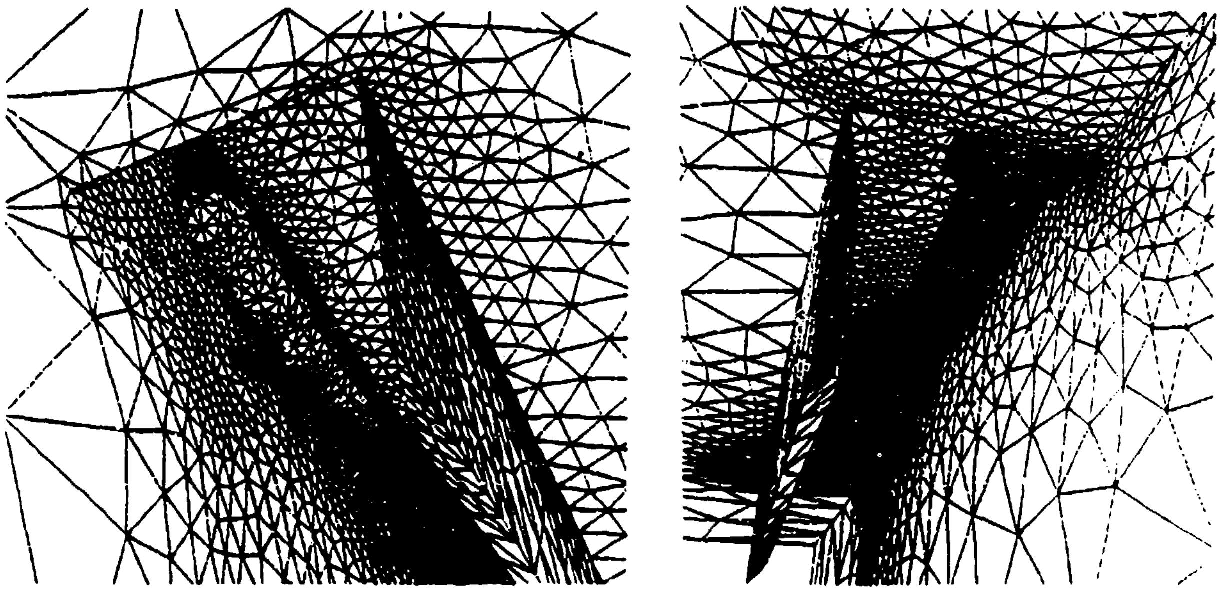
Shock Interaction with a T-62 Tank

- Grid Generator: FRGEN3D
- Hydro-Code: FEFLO74 (3-D, ALE-Euler, H-Ref.)
- Plotting: FEPLOT3D, FEPOST3D
- Details: [Ae.7, Ae.10]

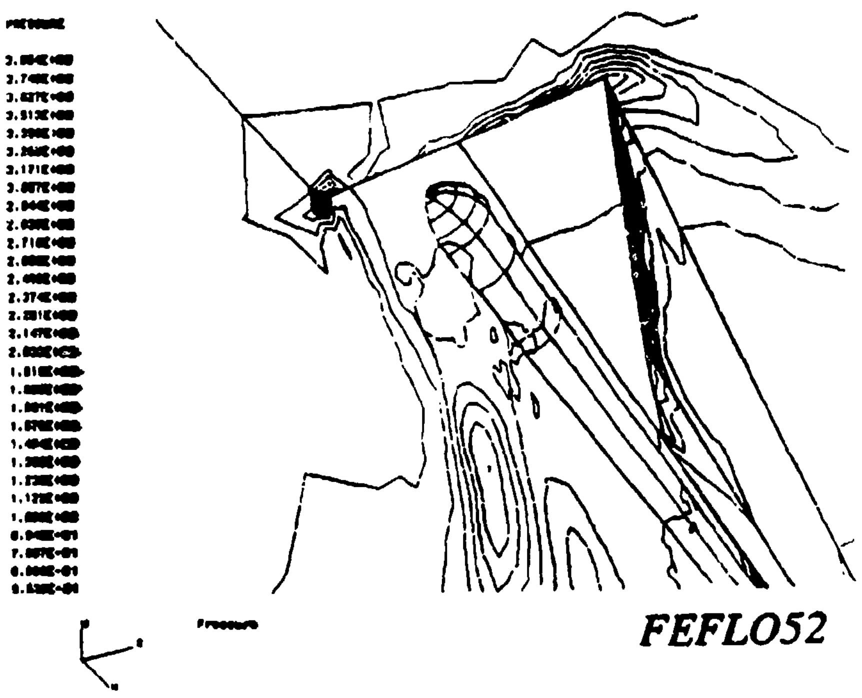


Examples 10 **1-Store Release into Supersonic Free Stream**

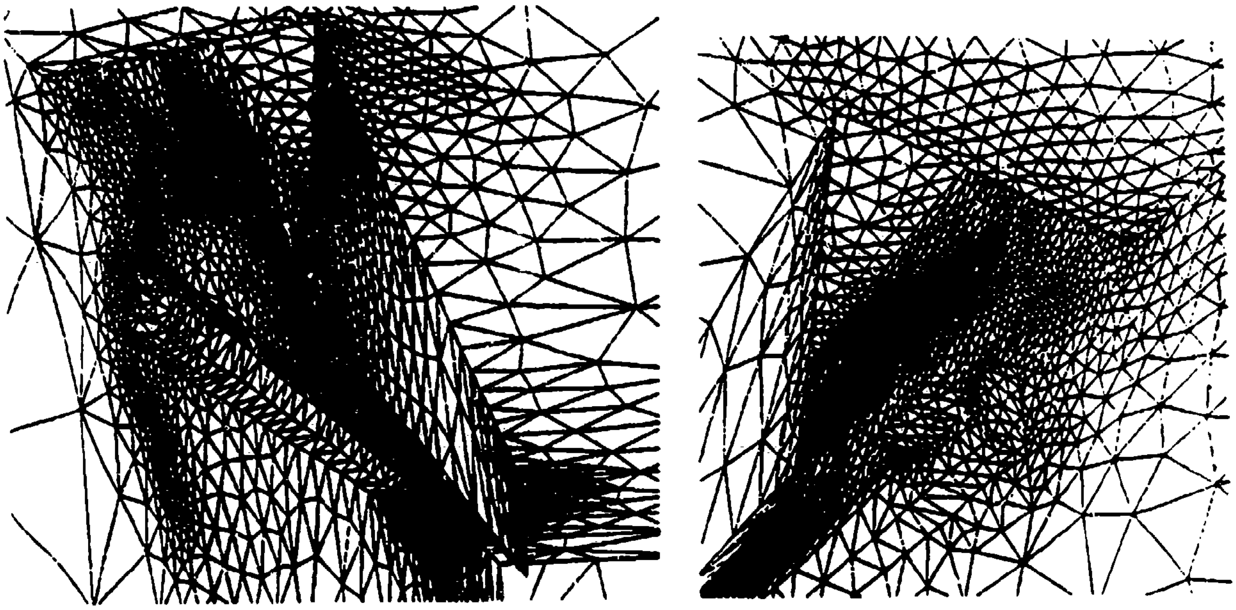
- Grid Generator: FRGEN3D
- Hydro-Code. FEFLO52 (3-D, ALE-Euler, Remeshing)
- Plotting: FEPLOT3D, FEPOST3D
- Details: [Ar.4]



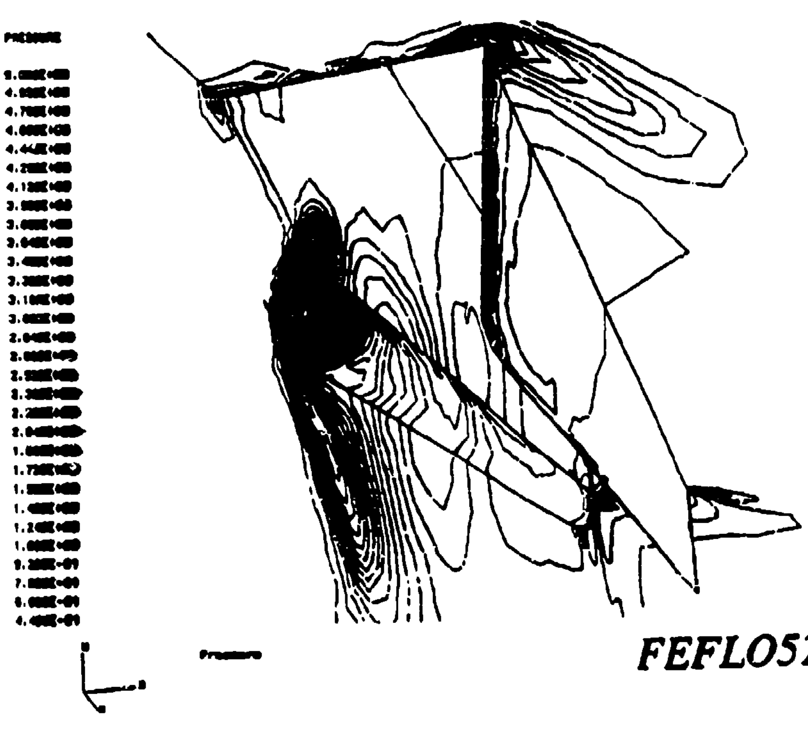
FEFLO52



FEFLO52



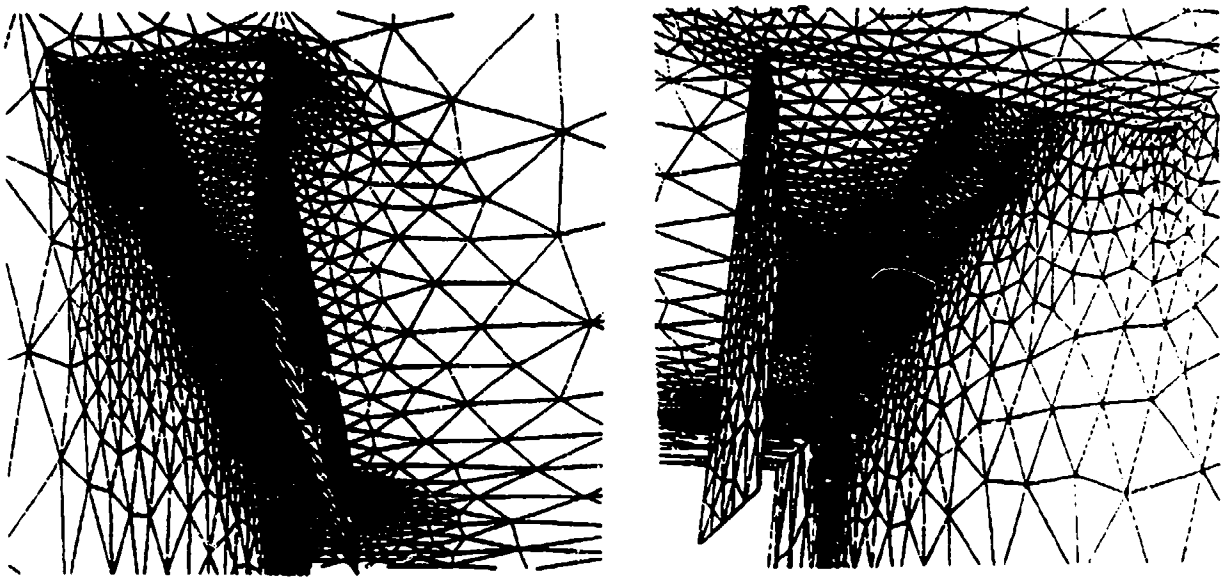
FEFLO52



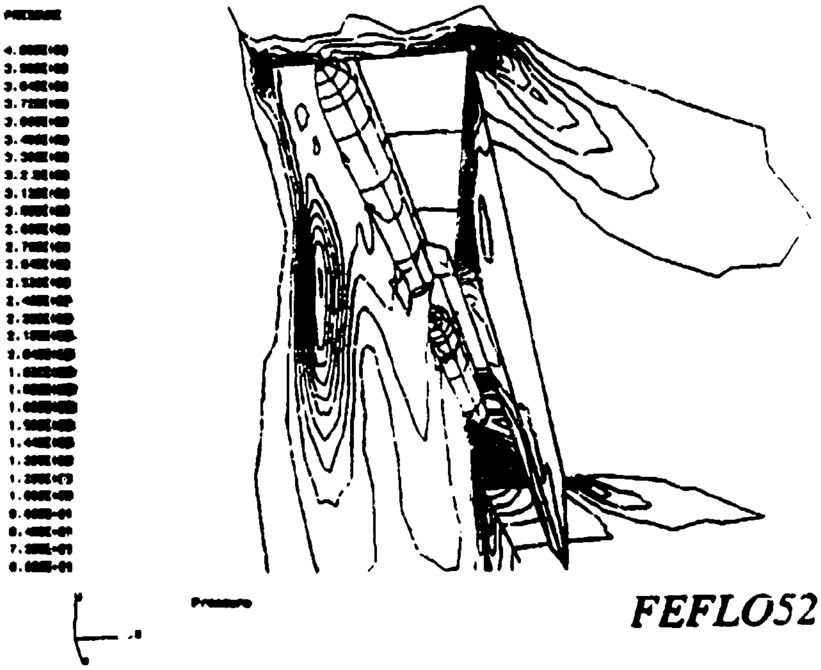
FEFLO52

Examples 11 **2-Store Release into Supersonic Free Stream**

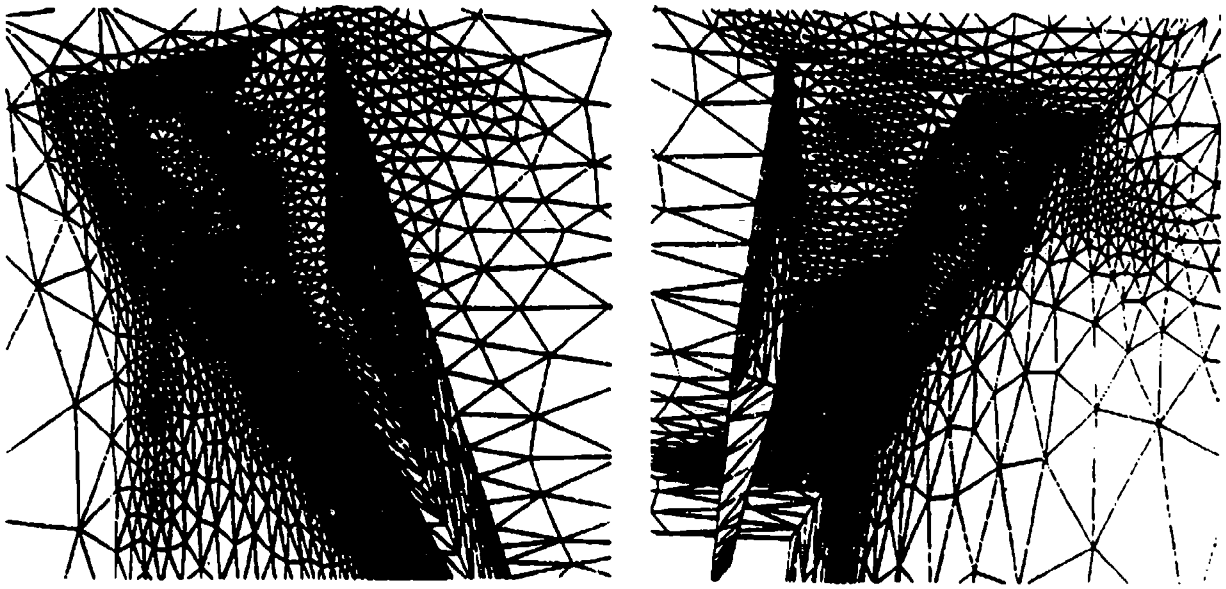
- Grid Generator: FRGEN3D
- Hydro-Code: FEFLO52 (3-D, ALE-Euler, Remeshing)
- Plotting: FEPLOT3D, FEPOST3D
- Details: [Ar.4]



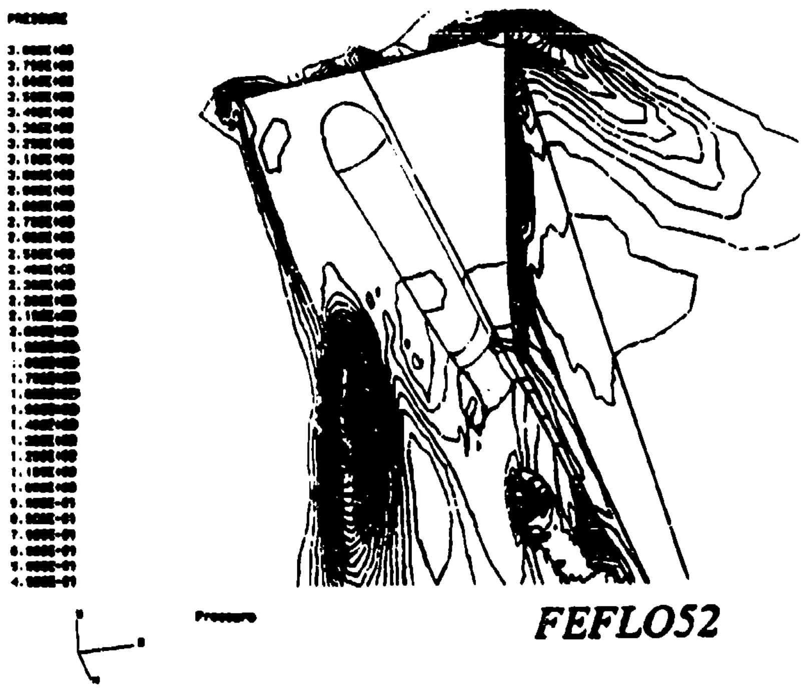
FEFLO52



FEFLO52



FEFLO52



FEFLO52

REFERENCES

OVERVIEWS
Books

B.1 O.C. Zienkiewicz and K. Morgan - *Finite Elements and Approximation*; J. Wiley & Sons. An excellent introductory book, especially for those with finite difference background.

B.2 O.C. Zienkiewicz - *The Finite Element Method*; McGraw Hill. The classic book on Finite Elements. Perhaps a bit heavy on examples from the Swansea team, but this only a minor flaw.

B.3 O. Pironneau - *Finite Element Methods for Fluids*; J. Wiley. A good overview of the field. Requires some background in Numerical Analysis.

EULER AND NAVIER-STOKES SOLVERS
Euler Solvers

Eu.1 A. Jameson, W. Schmidt and E. Turkel - Numerical Solution of the Euler Equations by Finite Volume Methods using Runge-Kutta Time-Stepping Schemes; AIAA-Paper 81-1259 (1981).

Eu.2 A. Jameson, T.J. Baker and N.P. Weatherhill - Calculation of Inviscid Transonic Flow over a Complete Aircraft; AIAA-86-0103 (1986).

Eu.3 P.D. Lax and B. Wendroff - Systems of Conservation Laws; *Comm.Pure Appl.Math.* 13, 217-237 (1960).

Eu.4 R.W. MacCormack - The Effect of Viscosity in Hypervelocity Impact Cratering; AIAA 69-354 (1969).

Eu.5 J. Donea - A Taylor Galerkin Method for Convective Transport Problems; *Int.J.Num.Meth.Engng.* 20, 101-119 (1984).

Eu.6 R. Löhner, K. Morgan and O.C. Zienkiewicz - The Solution of Nonlinear Systems of Hyperbolic Equations by the Finite Element Method; *Int.J.Num.Meth.Fluids* 4, 1043-1063 (1984).

Eu.7 R. Löhner, K. Morgan, J. Peraire and O.C. Zienkiewicz - Finite Element Methods for High Speed Flows; AIAA-85-1531-CP (1985).

FCT

Fc.1 J.P. Boris and D.L. Book - Flux-corrected Transport. I. SHASTA, a Transport Algorithm that works; *J.Comp.Phys.* 11, 38- (1973).

Fc.2 D.L. Book, J.P. Boris and K. Hain - Flux-corrected Transport. II. Generalizations of the Method; *J.Comp.Phys.* 18, 248- (1975).

Fc.3 J.P. Boris and D.L. Book - Flux-corrected Transport. III. Minimal-Error FCT Algorithms; *J.Comp.Phys.* 20, 397-431 (1976).

Fc.4 S.T. Zalesak - Fully Multidimensional Flux-Corrected Transport Algorithm for Fluids; *J.Comp.Phys.* 31, 335-362 (1979).

Fc.5 A.K. Parrott and M.A. Christie - FCT Applied to the 2-D Finite Element Solution of Tracer Transport by Single Phase Flow in a Porous Medium; Proc. ICFD-Conf. on Numerical Methods in Fluid Dynamics, Reading, Academic Press (1986).

Fc.6 R. Löhner, K. Morgan, J. Peraire and M. Vahdati - Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations; ICASE Rep. 87-4, *Int.J.Num.Meth. Fluids* 7, 1093-1109 (1987).

Fc.7 R. Löhner, K. Morgan, M. Vahdati, J.P. Boris and D.L. Book - FEM-FCT: Combining Unstructured Grids with High Resolution; *Comm. Appl. Num. Meth.* 4, 717-730 (1988).

TVD Schemes

Tv.1 P. Woodward and P. Colella - The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks; *J.Comp.Phys.* 54, 115-173 (1984).

Tv.2 P. Colella - Multidimensional Upwind Methods for Hyperbolic Conservation Laws; LBL-17023, Preprint (1983).

Tv.3 P.K. Sweby - High Resolution Schemes Using Flux Limiters for Hyperbolic Conservation Laws; *SIAM J.Num.Anal.* 21, 995-1011 (1984).

Tv.4 P.L. Roe - Approximate Riemann Solvers, Parameter Vectors and Difference Schemes; *J.Comp.Phys.* 43, 357-372 (1981).

Tv.5 B. van Leer - Towards the Ultimate Conservative Scheme. II. Monotonicity and Conservation Combined in a Second Order Scheme; *J.Comp.Phys.* 14, 361-370 (1974).

Tv.6 A. Harten - High Resolution Schemes for Hyperbolic Conservation Laws; *J.Comp.Phys.* 49, 357-393 (1983).

Tv.7 S. Osher and F. Solomon - Upwind Difference Schemes for Hyperbolic Systems of Conservation Laws; *Math.Comp.* 38, 339-374 (1982).

Tv.8 D.L Whitaker, B. Grossman and R. Löhner - Two-Dimensional Euler Computations on a Triangular Mesh Using an Upwind, Finite-Volume Scheme; AIAA-89-0365 (1989).

Unstructured Multigrid Methods

Mg.1 R. Löhner and K. Morgan - An Unstructured Multigrid Method for Elliptic Problems; *In-*

t.J.Num.Meth.Eng. 24, 101-115 (1987).

Mg.2 R. Löhner and K. Morgan - Unstructured Multigrid Methods; Proc. of the Second European Multigrid Conf. (U. Trottenberg and W. Hackbusch eds.), GMD-Studien Nr 110, GMD, Sankt Augustin (1986).

Mg.3 W. Gentzsch and A. Schlüter - Über ein Einschrittverfahren mit zyklischer Schrittweitenänderung zur Lösung parabolischer Differentialgleichungen; *ZAMM* 58, T415-T416 (1978).

Mg.4 E. Perez, J. Periaux, J.P. Rosenblum, B. Stoufflet, A. Dervieux and M.H. Lallemand - Adaptive Full-Multigrid Finite Element Methods for Solving the Two-Dimensional Euler Equations; Proc. 10th Int.Conf.Num.Meth. Fluid Dynamics, Peking, June 1986. Springer Verlag.

Mg.5 D.J. Mavriplis and A. Jameson - Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes; *AIAA* J. 28, 8, 1415-1425 (1990).

Snakes and Linelets

Sl.1 O. Hassan, K. Morgan, and J. Peraire - An Implicit Finite Element Method for High Speed Flows. AIAA-90-0402 (1990).

Sl.2 R. Löhner and D. Martin - An Implicit, Linelet-Based Solver for Incompressible Flows; pp. 9-20 in *Advances in Finite Element Analysis in Fluid Dynamics* (M.N. Dhaubhadel, M.S. Engelman and J.N. Reddy eds.), FED-Vol. 123, ASME (1991).

MESH ADAPTATION

Mesh Adaptation by Movement

Am.1 P. A. Gnoffo - A Finite-Volume, Adaptive Grid Algorithm Applied to Planetary Entry Flowfields; *AIAA J.* 21, 1249-1254 (1983).

Am.2 A.R. Diaz, N. Kikuchi and J.E. Taylor - A Method for Grid Optimization for the Finite Element Method; *Comp.Meth.Appl.Mech.Eng.* 41, 29-45 (1983).

Am.3 K. Nakahashi and G. S. Deiwert - A Three-Dimensional Adaptive Grid Method; AIAA-85-0486 (1985).

Am.4 B. Palmerio and A. Dervieux - Application of a FEM Moving Node Adaptive Method to Accurate Shock Capturing; Proc. First Int. Conf. on Numerical Grid Generation in CFD, Landshut, W. Germany, July 14-17 (1986).

Mesh Adaptation by Enrichment

Ae.1 W. Schönauer, K. Raith and K. Glotz - The Principle of Difference Quotients as a Key to the Self-Adaptive Solution of Nonlinear Partial Differential Equations; *Comp.Meth.Appl.Mech.Eng.* 28, 327-359 (1981)

Ae.2 M.J. Berger and J. Oliger - Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations; *J.Comp.Phys.* 53,484-512 (1984).

Ae.3 J.F. Dannenhoffer and J.R. Baron - Robust Grid adaptation for Complex Transonic Flows; AIAA-86-0495 (1986).

Ae.4 B. Palmerio, V. Billey, A. Dervieux and J. Periaux - Self-Adaptive Mesh Refinements and Finite Element Methods for Solving the Euler Equations; Proc. ICFD Conf. on Numerical Methods for Fluid Dynamics, Reading, U.K., March 1985.

Ae.5 J.T. Oden, P. Devloo and T. Strouboulis - Adaptive Finite Element Methods for the Analysis of Inviscid Compressible Flow: I. Fast Refinement/Unrefinement and Moving Mesh Methods for Unstructured Meshes; *Comp. Meth. Appl. Mech. Eng.* 59, 327-362 (1986).

Ae.6 R. Löhner - An Adaptive Finite Element Scheme for Transient Problems in CFD; *Comp.Meth.Appl.Mech.Eng.* 61, 323-338 (1987).

Ae.7 R. Löhner - Adaptive H-Refinement on 3-D Unstructured Grids for Transient Problems; *AIAA*-89-0365 (1989).

Ae.8 J.D. Baum and R. Löhner - Numerical Simulation of Shock-Elevated Box Interaction Using an Adaptive Finite Element Shock Capturing Scheme; AIAA-89-0653 (1989).

Ae.9 R. Löhner and J.D. Baum - Numerical Simulation of Shock Interaction with Complex Geometry Three-Dimensional Structures using a New Adaptive H-Refinement Scheme on Unstructured Grids; AIAA-90-0700 (1990).

Ae.10 J.D. Baum and R. Löhner - Numerical Simulation of Shock Interaction with a Modern Main Battlefield Tank; *AIAA*-91-1666 (1991).

Ae.11 D.J. Mavriplis - Unstructured and Adaptive Mesh Generation for High Reynolds-Number Viscous Flows; pp. 79-91 in Proc. 3rd Int. Conf. Num. Grid Gen. in CFD (Sengupta et al. eds), Pineridge Press (1988).

Ae.12 D.J. Mavriplis - Euler and Navier-Stokes Computations for Two-Dimensional Geometries Using Unstructured Meshes; *ICASE Rep.* 90-3 (1990).

Ae.13 O.C. Zienkiewicz, J.P. de S.R. Gago and D.W. Kelly - The Hierarchical Concept in Finite Element Analysis; Comp.Struct. 16, 53-65 (1983).

Ae.14 I. Babuska, J. Chandra and J.E. Flaherty (eds.) - Adaptive Computational Methods for Partial Differential Equations; SIAM Philadelphia (1983).

Mesh Adaptation by Remeshing

Ar.1 J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz - Adaptive Remeshing for Compressible Flow Computations; *J. Comp. Phys.* 72, 449-466 (1987).

Ar.2 R. Löhner - Adaptive Remeshing for Transient Problems; *Comp. Meth. Appl. Mech. Eng.* 75, 195-214 (1989).

Ar.3 R. Löhner - An Adaptive Finite Element Solver for Transient Problems with Moving Bodies; *Comp.Struct.* 30, 303-317 (1988).

Ar.4 R. Löhner - Three-Dimensional Fluid-Structure Interaction Using a Finite Element Solver and Adaptive Remeshing; *Computer Systems in Engineering* 1, 2-4, 257-272 (1990).

Ar.5 R. Tilch - PhD Thesis, CERFACS (1991).

GRID GENERATION

Books

Gb.1 J.F. Thompson, Z.U.A. Warsi and C.W. Mastin - *Numerical Grid Generation: Foundations and Applications*; North Holland (1985).

Gb.2 P.L. George - *Automatic Mesh Generation*; J. Wiley & Sons (1991).

Grid Generation: Domain Definition by Digitization

Gd.1 M. Merriam and T. Barth - 3D CFD in a Day: The Laser Digitizer Project; AIAA-91-1654 (1991).

Grid Generation: Advancing Front Approach

Ga.1 N. van Phai - Automatic Mesh generation with Tetrahedron Elements; *Int.J.Num.Meth.Eng.* 18, 237-289 (1982).

Ga.2 S.H. Lo - A New Mesh Generation Scheme for Arbitrary Planar Domains; *Int.J.Num.Meth.Eng.* 21, 1403-1426 (1985).

Ga.3 J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz - Adaptive Remeshing for Compressible Flow Computations; *J. Comp. Phys.* 72, 449-466 (1987).

Ga.4 R. Löhner - Some Useful Data Structures for the Generation of Unstructured Grids; *Comm.Appl.Num.Meth.* 4, 123-135 (1988).

Ga.5 R. Löhner and P. Parikh - Three-Dimensional Grid Generation by the Advancing Front Method; *Int.J.Num.Meth. Fluids* 8, 1135-1149 (1988).

Ga.6 J. Peraire, K. Morgan, and J. Peiro - Unstructured Finite Element Mesh Generation and Adaptive Procedures for CFD. *AGARD-CP-464*, 18 (1990).

Ga.7 R. Löhner, J. Camberos and M. Merriam - Parallel Unstructured Grid Generation; to appear in *Comp.Meth.Appl.Mech.Eng.* (1992).

Ga.8 F. Huet - Generation de Maillage Automatique dans les Configurations Tridimensionelles Complexes Utilization d'une Methode de 'Front'; AGARD-CP-464, 17 (1990).

Grid Generation: Voronoi Approach

Gv.1 J.C. Cavendish - Automatic Triangulation of Arbitrary Planar Domains for the Finite Element Method; *Int.J.Num.Meth.Eng.* 8, 679-696 (1974).

Gv.2 D.T. Lee and B.J. Schachter - Two Algorithms for Constructing a Delaunay Triangulation; *Int.J.Comp.Inf.Sc.* 9,3, 219-242 (1980).

Gv.3 A. Bowyer - Computing Dirichlet Tesselations; *The Computer Journal* 24,2, 162-167 (1981).

Gv.4 D.F. Watson - Computing the N-Dimensional Delaunay Tesselation with Application to Voronoi Polytopes; *The Computer Journal* 24,2, 167-172 (1981).

Gv.5 M. Tanemura, T. Ogawa and N. Ogita - A New Algorithm for Three- Dimensional Voronoi Tesselation; *J.Comp.Phys.* 51, 191-207 (1983).

Gv.6 S.W. Sloan and G.T. Houlsby - An Implementation of Watson's Algorithm for Computing 2-Dimensional Delaunay Triangulations; *Adv.Eng.Software* 6,4, 192-197 (1984).

Gv.7 R.C. Kirkpatrick - Nearest Neighbor Algorithm; pp.302-309 in Springer Lecture Notes in Physics 238 (M.J. Fritts, W.P. Crowley and H. Trease eds.), Springer Verlag (1985).

Gv.8 D.N. Shenton and Z.J. Cendes - Three-Dimensional Finite Element Mesh Generation Using Delaunay Tesselation; IEEE Trans. on Magnetics, MAG-21, 2535-2538 (1985).

Gv.9 J.C. Cavendish, D.A. Field and W.H. Frey - An Approach to Automatic Three-Dimensional Finite Element Generation; *Int. J. Num. Meth. Eng.* 21, 329-347 (1985).

Gv.10 A. Jameson, T.J. Baker and N.P. Weatherhill - Calculation of Inviscid Transonic Flow over a Complete Aircraft; AIAA-86-0103 (1986).

Gv.11 D.G. Holmes and D.D. Snyder - The Generation of Unstructured Triangular Meshes Using Delaunay Triangulation; pp. 643-652 in Numerical Grid Generation in Computational Fluid Dynamics (Sengupta et al. eds.), Pineridge Press (1988).

Gv.12 T.J. Baker - Developments and Trends in Three-Dimensional Mesh Generation. *Appl. Num. Math.* 5, 275-304 (1989).

Grid Generation: Modified Octree Approach

Go.1 M.A. Yerry and M.S. Shepard - Automatic Three-Dimensional Mesh Generation by the Modified-Octree Technique; *Int.J.Num.Meth.Eng.* 20, 1965-1990 (1984).

Grid Generation: Macro Element Approach

Gm.1 O.C. Zienkiewicz and D.V. Phillips - An Automatic Mesh Generation Scheme for Plane and Curved Surfaces by Isoparametric Co-ordinates; *Int.J.Num.Meth.Eng.* 3, 519-528 (1971).

Gm.2 J.F. Thompson - A Composite Grid Generation Code for General 3S Regions - The EAGLE Code; *AIAA J.* 26, 3, 271ff (1988).

Gm.3 J.P. Steinbrenner, J.R. Chawner and C.L. Fouts - A Structured Approach to Interactive Multiple Block Grid Generation; *AGARD-CP-464*, 8 (1990).

Gm.5 S. Allwright - Multiblock Topology Specification and Grid Generation for Complete Aircraft Configurations; *AGARD-CP-464*, 11 (1990).

Grid Generation: From Regular Background Grid

Gr.1 W.C. Thacker, A. Gonzalez and G.E. Putland - A Method for Automating the Construction of Irregular Computational Grids for Storm Surge Forecast Models; *J.Comp.Phys.* 37, 371-387 (1980).

Gr.2 D.G. Holmes and S.C. Lamson - Compressible Flow Solutions on Adaptive Triangular Meshes; Open Forum AIAA - Reno'86 - Meeting (1986).

Data Structures, Algorithms and Programming Techniques

Ds.1 D. E. Knuth, "The Art of Computer Programming," Vols. 1-3 (Addison-Wesley, Reading, MA, 1973).

Ds.2 R. Sedgewick - *Algorithms* ; Addison-Wesley (1983).

Ds.3 J.W.J. Williams - Heapsort; *Comm.ACM* 7, 347-348 (1964).

Ds.4 R.W. Floyd - Treesort 3; *Comm.ACM* 7, 701 (1964).

EFFECTIVE USE OF SUPERCOMPUTER HARDWARE

Scatter-add

Sc.1 R. Dieckkämper - Vectorized Finite Element Analysis of Nonlinear Problems in Structural Dynamics; pp.293-298 in Proc Parallel Computing '83 (M. Feilmeier, G. Joubert and U. Schendel eds.), North Holland (1984).

Sc.2 R. Löhner and K. Morgan - Finite Element Methods on Supercomputers: The Scatter Problem; Proc. NUMETA'85 Conf. (J. Middleton and G. Pande eds.), 987-990, A.A. Balkema, Rotterdam, 1985.

Sc.3 SINDA Thermal Analyzer Manual (1980)

Sc.4 D.J. Mavriplis - Three-Dimensional Unstructured Multigrid for the Euler Equations; AIAA-91-1549 (1991).

Sc.5 J. Peraire, J. Peiro and K. Morgan - A 3-D Finite Element Multigrid Solver for the Euler Equations; AIAA-92-0449 (1992).

Connection Machine: Issues and Apllications

Cm.1 CM routers NEED REF

Cm.2 CM renumbering schemes NEED REF

Cm.3 D. Jespersen and C. Levit - A Computational Fluid Dynamics Algorithm on a Massively Parallel Machine; AIAA-89-1936-CP (1989).

Cm.4 L.N. Long, M.M.S. Khan and H.T. Sharp - A Massively Parallel Three-Dimensional Euler/Navier-Stokes Method; AIAA-89-1937-CP (1989).

Cm.5 E.S. Oran, J.P. Boris, and E.F. Brown - Fluid-Dynamic Computations on a Connection Machine - Preliminary Timings and Complex Boundary Conditions; AIAA-90-0335 (1990).

Cm.6 CM application NEED REF

MIMD Machines: Issues and Apllications

Mi.1 H. Simon - Partitioning of Unstructured Mesh Problems for Parallel Processing; Proc. Conf. Parallel Meth. on Large Scale Structural Analysis and Physics Appl., Pergamon Press (1991).

A Frontal Approach for Node Generation in Delaunay Triangulations.

J.-D. Müller [*]
P.L. Roe [†]
H. Deconinck [‡]
CFD-Group - von Karman Institute for Fluid Dynamics
Aerospace Department - University of Michigan

1 SUMMARY

A new algorithm for the generation of the interior nodes for Delaunay triangulations is given. The method uses a background grid to interpolate local mesh size parameters that is taken from the triangulation of the given boundary nodes . Geometric criteria are used to find a set of nodes in a frontal manner. This set is subsequently introduced into the existing mesh, thus providing an updated Delaunay triangulation. The procedure is completed when no more improvement of the grid by inserting new nodes can be achieved.

2 INTRODUCTION

The purpose of this lecture series is to discuss the achievements and recent advances made in the solution of the Euler- and Navier-Stokes equations on unstructured grids. One of the main benefits related to the unstructured grid approach is the simplicity with which an unstructured grid can be tailored around very complex geometries and be adapted to the solution in a subsequent interaction process with the flow solver. Such advantage will become even more important as three-dimensional calculations become commonplace.

It is desirable that the mesh can be generated with minimum input from the user. Ideally, one would wish to specify only the boundary geometry, and perhaps a function to specify some desired mesh size. The internal point cloud should then be found automatically by the grid generation code. Two methods that satisfy these criteria have become popular and are discussed in other lecture series contributions :

- The Advancing Front method of Peraire et al. [1], detailed in the contributions of Lohner and Morgan & Peraire.
- Holmes' refinement procedure [2] used in a Delaunay triangulation [3] as discussed in the lectures by Barth.

The approach we propose here combines ideas from both methods.

In itself Delaunay triangulation does not provide an interior point cloud. A popular approach to overcome this deficiency has been outlined by Holmes [2]. A Delaunay triangulation of the boundary nodes is taken as an initial grid. Figure 1 gives such a triangulation for the three element aerofoil configuration shown in figure 11. The initial triangulation consists of large, very skewed triangles that are found to exceed certain thresholds of maximum area or maximum skewness. Holmes proposes to measure skewness as the ratio of the radius of the circumscribed circle over twice the radius of the inscribed circle.

Once such a bad triangle is detected it will be refined by the insertion of a new node at the circumcentre of that triangle. Refinement is performed on the largest triangle in the grid until all triangles are smaller than a first threshold value. Then refinement continues on the skewed triangles starting with the one having the largest circumcircle. The final grid is obtained after all skewed triangles are smaller than a second area threshold. Refining on skewness yields an implicit mechanism that increases the node density very close to boundaries with a finer node spacing.

However, searching for the largest cell or a skewed cell with the largest circumcircle for each new node is a rather costly procedure and the skewness criterion is expensive to evaluate since it involves three square roots during the calculation of the circle ratio. Also the grids produced by this approach are not as regular as the ones given by the Advancing Front method. This is due to the fact that the refinement process is much more random.

3 FRONTAL NODE CREATION

We introduce here a technique that combines the ideas of refining a Delaunay triangulation of boundary nodes with the ideas of frontal advance. In our method, the front will take the form of a boundary between a 'nicely' triangulated region and a 'badly' triangulated region. The method will be described for two dimensions, but we believe there will be few obstacles to implementing it in three.

If one looks at a Delaunay triangulation of a set of boundary nodes (as in figure 1), one can observe that almost every boundary face is either the short face of a triangle with one very acute angle or else one of two short faces in a triangle with one very obutse angle. In accordance with our idea of a front that divides nice triangles from bad triangles, we take the boundary to define the initial position of such a front. To begin with, we have no nice triangles, but we will introduce a layer of well-positioned nodes that will allow the front to advance.

The construction of the new nodes is an easy task. We simply form an equilateral triangle with the frontal face and either stretch or compress it to better match the spacing requirements of the background mesh, this gives ideal spacing with the two nodes that form the face. A further check is required to make sure the new node is sufficiently distant from the remaining nodes in the grid and from the other new nodes. The desired distance is in-

[*]von Karman Institute and University of Michigan
[†]University of Michigan
[‡]von Karman Institute

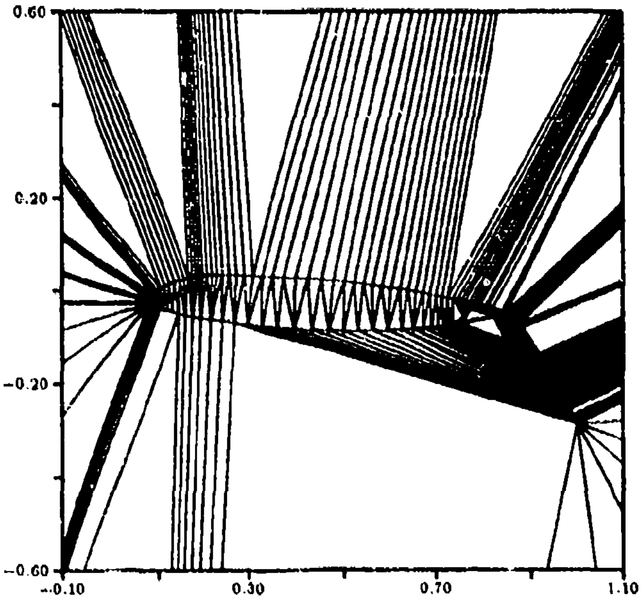


Fig. 1: Triangulation of boundary nodes. The nodes on the boundary of a three element aerofoil are connected to nodes on the outer boundary (not shown), nodes on another boundary or nodes on the same boundary.

terpolated on a background mesh that uniquely specifies the distance between nodes everywhere in the domain. Nodes that exhibit bad spacing are either merged with other nodes or discarded. With these new nodes in place, the Delaunay algorithm is re-run and will readily accept the proposed, nice triangles as it resents skewness in its triangulation. We will name these triangles 'explicit' in the following. Also nice triangles between the new nodes will be formed as they are guaranteed to be spaced nicely as well, the 'implicit' triangles. In the rest of the domain Delaunay still has to construct acute cells, though with slightly improved shape.

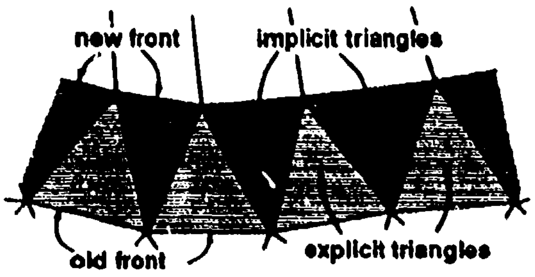


Fig. 2: Explicit triangles (striped) and implicit triangles (squared) that are formed along the old front and build the new front.

Again the short faces of these acute cells denote a frontier between the region with nice cells and the region still waiting to be refined figure 2, and the process can be repeated until all bad triangles have vanished. Hence, the algorithm can be cast into the following steps:

1. detect all bad triangles in the grid and find their short faces,
2. find a set of nodes to form nice triangles with the short faces,
3. check whether the new nodes are not too close to any other node already introduced into the structure,
4. check whether the new nodes are not too close to any other new node,
5. retriangulate with the set of new nodes.

The steps will be repeated until no more improvement by node insertion can be achieved.

3.1 Front Detection.

The front consists of the interface between the region of properly refined triangles and the unrefined region. A refinement should only take place on a face that has a refinable triangle on one side and an unrefinable on the other. If refining was merely based on side ratios, an obtuse triangle in the front would lead to the introduction of three nodes. Figure 3 shows the two nodes that would be formed from the two short faces in the front of the obtuse triangle and the node from the face of the acute triangle that neighbors the obtuse one. Not that this extra node is badly placed, but this node should be formed only in the next stage. This would lead to an irregular front with scattered faces that may not be connnected and the subsequent refining would have much of the randomness of Holmes' method.

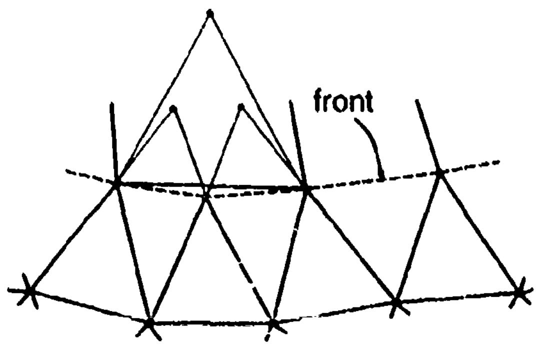


Fig. 3: Obtuse triangle along the front with three new nodes formed.

A triangle is unrefinable if either it is not skewed or it is skewed but node spacing around the cell does not allow further insertion. Checking is simplified by keeping a status flag for each triangle to only examine each cell once. It is to be emphasized that contrary to the usual Advancing Front method no explicit tracking of the front and thus no expensive overhead is required.

3.2 Node Construction.

The ideal node to be placed in the mesh would satisfy the distance criterion with all neighboring nodes, i.e. the distance to all nodes that it will be connected to equals the background spacing evaluated at the midpoint between these two. Clearly, this is an ill-posed problem. But even trying to satisfy the distance condition with the two nodes of the frontal face leads to a system of two quadratic equations. The task will become more amenable with the restriction to isocelеs triangles. We will carry out the geometrical construction in an approximate manner leading to only one linear equation.

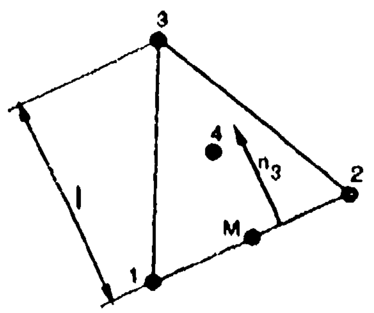


Fig. 4: Short face and constructed node. The new node 3 and the spacing reference node 4 lie on the perpendicular bisector of the face 3.

We approximate the length of the sides $l1$ and $l2$ opposite nodes 1 and 2 by $2/\sqrt{3}l$ of the height l as found in an equilateral triangle. Requiring that this approximated sidelength equals the desired spacing h_4 midway between node 3 and the midpoint of the base M (figure 4), we find

$$h_4 = \frac{2}{\sqrt{3}}|\mathbf{x}_3 - \mathbf{x}_M| = \left(\frac{1}{2}(\mathbf{x}_3 - \mathbf{x}_M)\nabla h + h_M\right)$$

where h_M denotes the desired spacing at M, ∇h is the local gradient of the background spacing and $\mathbf{x}_3,\mathbf{x}_M$ are the position vectors of nodes 3 and M. As we place the new node along the perpendicular bisector, we can write

$$\mathbf{x}_3 - \mathbf{x}_M = l\frac{\mathbf{x}_3 - \mathbf{x}_M}{|\mathbf{x}_3 - \mathbf{x}_M|} = l\mathbf{n}_3,$$

where $\mathbf{n}_3$ is the unit normal on the base pointing towards the triangle to be refined. The height for a triangle with counterclockwise sense is thus

$$l = \frac{h_M}{\frac{2}{\sqrt{3}} - \frac{1}{2}\mathbf{n}_3\nabla h}$$

Note that in the given form the altitude of the explicit triangle is independent of the length of the base. This conserves the thickness of the layer of cells introduced even if the length of the faces varies strongly (figure 9).

3.3 Searching.

The efficiency of unstructured grid generation methods is very dependent on the way specific nodes or cells are found in the grid. For example Bowyer's algorithm requires the search for a circumcircle that contains a new node and interpolation on a background mesh involves finding the background cell that contains the point of interest. As already stated, an implementation of Bowyer's algorithm usually comes with the storage of the neighbors to each triangle and the position of its circumcentre. Hence, a straightforward way to locate a position in a Delaunay triangulation would be to walk along the Dirichlet tesselation from circumcentre to circumcentre closer towards the target. But this method does not necessarily converge as a Voronoi vertex can lie outside the its associated triangle.

A method of similar computational cost is to walk from cell to cell in the direction of the target. As we progress

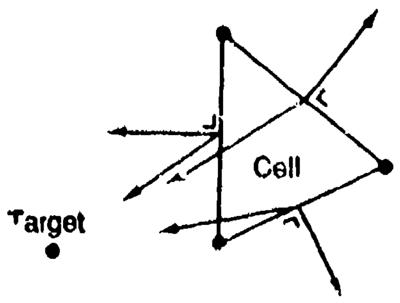


Fig. 5: Scalar product criterion to walk from a cell towards a target

at each step a finite distance towards our point of destiny we must reach our target so this search procedure always converges.

The direction to turn to is given by the maximum scalar product of the normal on the face and the vector from the midpoint of that face to the target (figure 5). Of course, only two directions have to be tested once the search is on its way, also it would be rather wasteful to use unit normals. The search is finished when all scalar products are non-positive, indicating that the target lies either in the cell or on a face of the cell.

In the worst case the cost of this search is $O(\sqrt{N})$ on a mesh with N nodes. However, once the foreground and background cells associated with the new node are determined, all of the triangles in the vicinity, where most of the remaining operations take place, are found in a few steps.

3.4 Background Mesh.

The Delaunay triangulation of all boundary nodes is computed as an initial triangulation to begin the point generation process. This triangulation provides at no extra cost a suitable background mesh to provide a local value of desired distance between nodes at any location within the convex hull. It will be assumed here that this desirable distance is a piecewise linear function of position, interpolated between the nodal values of a triangle in the background grid. The spacing value at that node is computed as the average distance to its two neighboring nodes on the boundary. The gradient of the spacing h is evaluated for each triangle in the background mesh using Gauss' Theorem,

$$\nabla h = \frac{1}{2S}\sum_{i=1}^{3} h_i \mathbf{s}_i,$$

where S stands for the surface and $\mathbf{s}_i$ for the scaled outward normals of the background cell.

A linear variation between the fine spacing on a body and the coarse spacing on a far-field boundary is obtained when the background triangle connects directly from the interior to the exterior boundary. But along concave contours it may happen that Delaunay connects between finely spaced interior boundaries and the background grid will specify a too large area of fine spacing. Figure 1 gives an example of such an illconnected background mesh. It shows a close-up of a multi element aerofoil, obscured by the triangles formed inside the elements. A clearer view of the configuration can be seen in figure 11. The triangles leaving the frame are connected to the outer boundary. However, the initial triangulation also connects the finely discretized trailing edge of the main flap with the lower side of the main aerofoil and implies an undesirable high node density in the entire illconnected area between the two elements.

The problem can easily be circumvented by the introduction of extra nodes into the background mesh. As the background triangulation consists of very skinny triangles, only very few nodes will be needed to break up the unwanted connectivity. If we connect these few nodes to a boundary in the background grid we implicitly define the spacing for these nodes in the same way as for the other boundaries. However, this procedure requires extra user input, usually after viewing the background grid and visualizing the grid becomes difficult in three dimensions. To be consistent with the philosophy of minimal user input we should have the program introduce the necessary nodes and merely ask the user which boundaries he does not want to have connected. The procedure will be to detect an illicit liaison and place a background node inbetween. During a subsequent retriangulation most if not all of the triangles shared between the two bodies will be broken and again few extra nodes suffice. Figure 6 shows the background mesh modified by automatic insertion. Four nodes have been introduced from triangles in the area between the main aerofoil and the main flap.

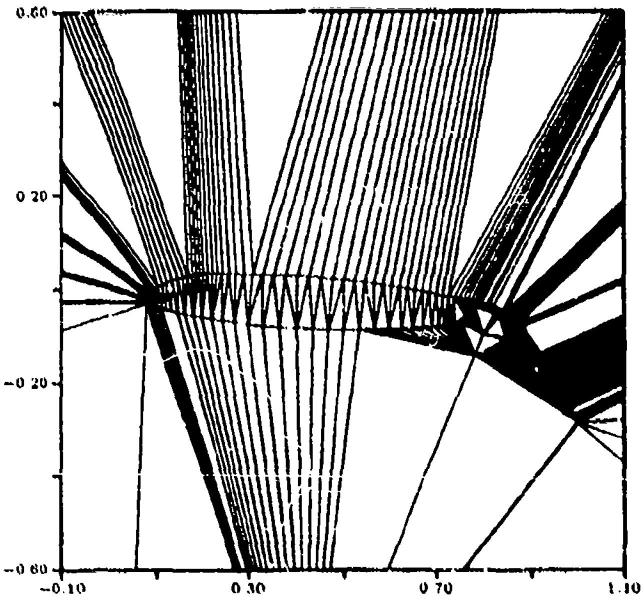


Fig. 6: Background grid automatically modified by the insertion of four nodes to break up unwanted connectivity Two of these nodes are shown between the main aerofoil and the second flap.

The remaining question is what spacing to apply to these new nodes. One would like the spacing to rise smoothly from every body node into the domain to finally match the spacing of the outer boundary. This corresponds to extrapolating the spacing with an average gradient from every boundary node towards the automatically inserted node and take the minimum of all these values – an unreasonably costly procedure.

Fortunately, the Delaunay properties make the task at hand a lot more amenable. If we place the new node to break an unwanted triangle at the Voronoi vertex of that triangle, we know that there is no node closer to the new node than the three nodes forming that triangle. Moreover, the new node is equidistant from both ill-connected boundaries. We then extrapolate from the spacing of the more finely discretized boundary using the average gradient of the initial triangulation.

3.5 Skewness Threshold.

So far the term 'bad' has been used for long skinny triangles without specifying on what we base this label. From the previous discussion it follows that a criterion is needed that is easy to evaluate and that discriminates the faces to be used in the front. An obvious and inexpensive choice for quantifying the proportions of a triangle is to look at ratios of the squared sidelength over the squared maximum sidelength. A triangle will be called 'bad' once any of its three side ratios drops below a threshold. Considering the fact that a triangle is formed by placing a node somewhere along the perpendicular bisector of the base, one can estimate threshold values for the side ratios. A first estimate can be derived for an acute triangle on a zero-gradient background. In this case Delaunay triangulation will always form an equilateral explicit triangle with the base, and an implicit triangle with the new node and the distant node of the previous bad triangle.

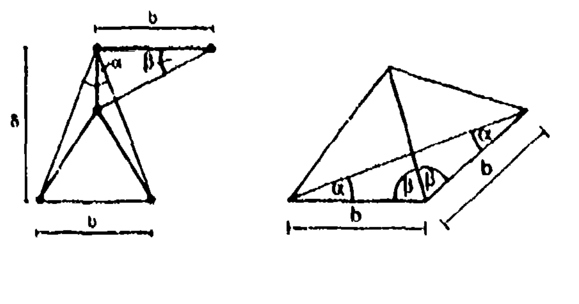


Fig. 7: (a) Refining of an acute triangle (dashed) into an equilateral triangle (bottom) and an acute 'implicit' triangle (top); (b) Refining of an obtuse triangle (dashed) into two triangles (full).

Figure 7 (a) shows the geometry in question. The worst 'implicit' triangle is produced when the distant node of the acute triangle also lies on the perpendicular bisector. If we let the dashed triangle become less and less acute α will grow and β will become smaller. Both angles will be equal if $\delta \approx .646$, so that refining an acute triangle with $\delta < .646$ will make the grid worse. Hence for acute triangles a good threshold is the side ratio of a triangle with $\delta = .646$ or

$$\left(\frac{S_{min}}{S_{max}}\right)^2 = .366.$$

Similar reasoning applies to the obtuse triangle in figure 7 (b). Here the two nodes formed perpendicular to the two short faces will be merged subsequently as they are too close to each other. Hence we have to consider the new node to be placed on the angular bisector. The smallest angle in the old triangulation was α; in the new triangulation it is β. Since $2(\alpha+\beta) = \pi$ matters only improve if $\alpha < \pi/4$. That is, we should only refine obtuse triangles for which the side ratio is less than

$$\left(\frac{S_{min}}{S_{max}}\right)^2 = .5.$$

It turns out that the quality of the triangulation is somewhat insensitive to the choice of the side ratio threshold and any value in the range of the two estimates will give good results. This allows to use the same threshold for both obtuse and acute triangles. The triangulation will change with a different threshold but the minimum angles found will remain virtually unchanged. Strong gradients in the background grid might lead to the formation of explicit triangles that exceed the threshold in the side ratio. Therefore the altitude of the triangle to be formed will be bounded to the height of an obtuse triangle with the threshold side length ratio and the height of its acute counterpart.

$$\sqrt{\left(\frac{S_{min}}{S_{max}}\right)^2 - \frac{1}{4}} \le \frac{l}{S_3} \le \sqrt{\left(\frac{S_{max}}{S_{min}}\right)^2 - \frac{1}{4}}$$

3.6 Spacing Check.

The spacing check balances the mechanism of node introduction due to excessive skewness by rejecting or merging nodes; in this way grid quality is assured for the implicit triangles. We might want to reject a new node because it falls too close tho some existing node, or because it falls too close to another new node. The two cases have to be dealt with separately.

Looking for close nodes that are already introduced, we can make use of the fact that a Delaunay mesh constructed with Bowyer's algorithm covers the entire convex hull. We can construct an enlarged circle around each triangle which is the circumcircle with an added rim of the required distance for the new node. If the new node does not fall within that enlarged circle, the distance between the new node and the nodes of the triangle is at least the required spacing (figure 8).

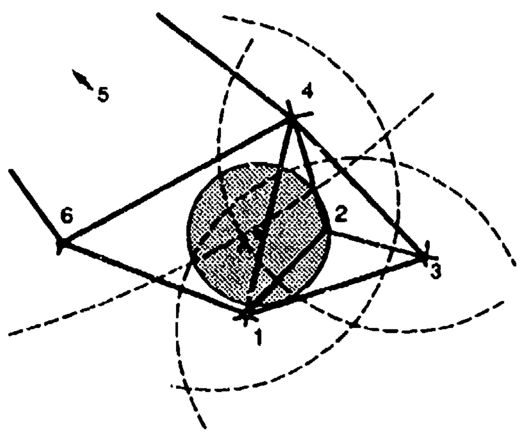


Fig. 8: Figure 8: A new node (A) is contained in the triangle 146, but closest to the node 2. The enlarged circles 124, 132, 342 contain A and require testing. The enlarged circle around 456 does not contain A and excludes nodes 4, 5 and 6 from testing.

In a similar fashion to the tree search during the insertion procedure, the simply connected region can be determined where nodes that are already introduced might be too close to a new node. Once a new node is found to be too close to another old one it is discarded from the list.

One is less fortunate with checking the distance towards the other new nodes that are also waiting to be introduced. Along the front we might find a set of very acute triangles that can lie rather oblique to it. The initial front along the boundaries in figure 1 can serve as a good example. New nodes that are too close to each other may not lie in neighboring triangles and one cannot make use of the underlying grid. Extensive search throughout the list of new nodes has to be performed, but the list contains only $O(\sqrt{N})$ nodes at a time. This advocates the use of an intelligent data structure that provides some kind of bucketing to further reduce the cost of searching and will retain an optimal count of operations of $O(N \log N)$. Once two new nodes are found to be too close, they are merged. This merging is actually the only step in the algorithm that introduces irregularity into an initially regular mesh. All other steps are independent of the order in which triangles or nodes are encountered. While it is generally not important which neighbors are merged, we do want to have preferred merging of the two nodes that are formed from the two short faces of an obtuse triangle as shown in figure 7 (b). If one searches backwards through the list of new nodes, this pair is encountered first and treated with higher priority.

In order to achieve large minimum angles we may tolerate nodes being closer to each other than allowed by the background mesh. Otherwise the skewness mechanism providing refinement may be counterbalanced too much. Initially the number of new nodes is completely determined by the number of boundary nodes. As the front propagates outwards, the nodes will eventually become too numerous as the background mesh demands more and more distance between the nodes and the spacing check will coarsen the front. In this way liberal spacing will allow more completely regular rows with the original number of nodes around the bodies. On the other hand, being too lax allows node insertion where no improvement can be achieved. Good values for a tolerance factor to be multiplied with the desired spacing have been found to lie in the range between .5 and .75.

It is to be noted that no criterion for too large cells is needed if the front emerges from the finely spaced boundaries. The spacing mechanism will gradually coarsen up that front until it meets the outer boundary. Further refinement in the field can be left to solution adaptive interaction with the solver. However, the implementation would not pose any problems. If, after retriangulation, two connected nodes are found to be too far apart, another node can be introduced between them.

3.7 Computational Cost.

Suppose that a total of N nodes are generated in such a way that N^p new nodes are created every time the front advances. Bowyer's algorithm takes $O(\log N)$ operations to introduce a single node into a triangulation. With a dissecting data structure like a split-tree the cost of searching the list of N^p new nodes requires $O(\log N^p)$ operations. One needs N^{1-p} stages to construct the full triangulation; the total cost is thus $O(N(p+1)\log N)$. As p ranges between 0 and 1, the number of operations necessary increases by a factor of two in the worst case when all nodes are formed in one single stage, compared to a triangulation of specified nodes. Hence, the method is asymptotically optimal.

Actual times are given for the three element aerofoil case given in figures 9 through 12. The initial triangulation of the 328 boundary nodes took .77 seconds on a DEC 5000/200, i.e, .0023 sec/node. The insertion algorithm created 2047 interior nodes and used 27 seconds, i.e .013 sec/node. Thus, the method for generating new nodes and triangulating them costs about five times as much computer time per node as the triangulation alone. Note that the current implementation does not employ any tree data structure and therefore the given times could be reduced further.

4 EXAMPLES

A classic case for an unstructured grid generator is the grid around a multi-element aerofoil. Structured grid generation already requires sophisticated extensions to deal with this problem. The background grid for the aerofoil given in figure 1 was modified by the automatic insertion of four nodes as seen in figure 6. The initial triangulation is boundary conformal and does not require introduction of extra nodes according Weatherill's aposteriori criterion [11].

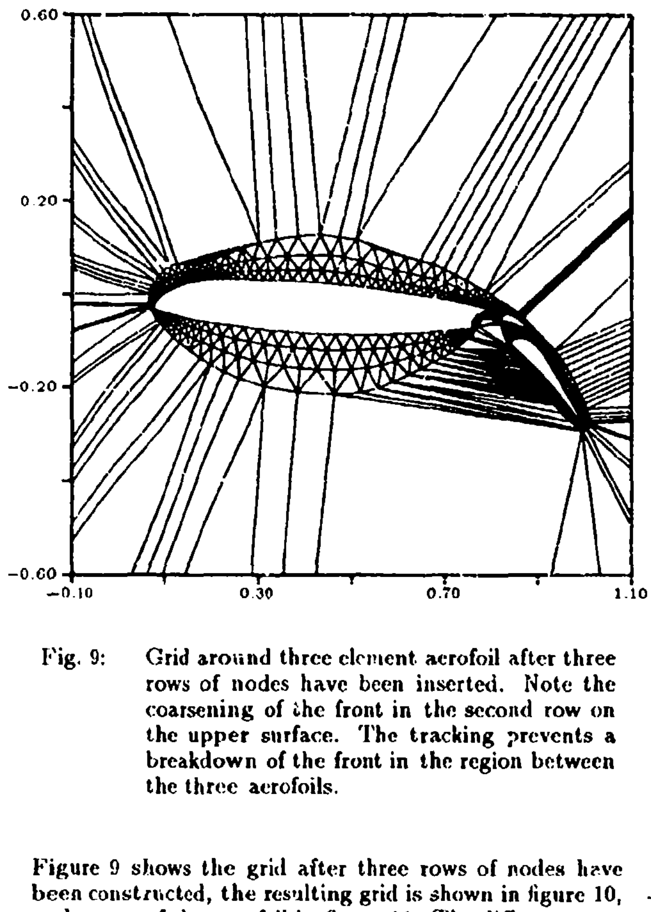


Fig. 9: Grid around three element aerofoil after three rows of nodes have been inserted. Note the coarsening of the front in the second row on the upper surface. The tracking prevents a breakdown of the front in the region between the three aerofoils.

Figure 9 shows the grid after three rows of nodes have been constructed, the resulting grid is shown in figure 10, a close-up of the aerofoil in figure 11. The different rows of nodes can be identified clearly in the final triangulation.

On the upper surface of the main aerofoil it can be demonstrated how essential the construction algorithm is to grid quality. In the second row the spacing check has eliminated several nodes and the length of the faces in the new front varies from normal to doubled. Still the nodes in the third row are aligned evenly, providing nearly equilateral triangles again. The disturbances introduced in the second row are completely eliminated in the fourth row.

The regularity of the grid is entirely due to the frontal insertion, no smoothing filter was applied. Figure 12 shows a detail of the grid between the three elements. The fronts do not break down and merge into each other

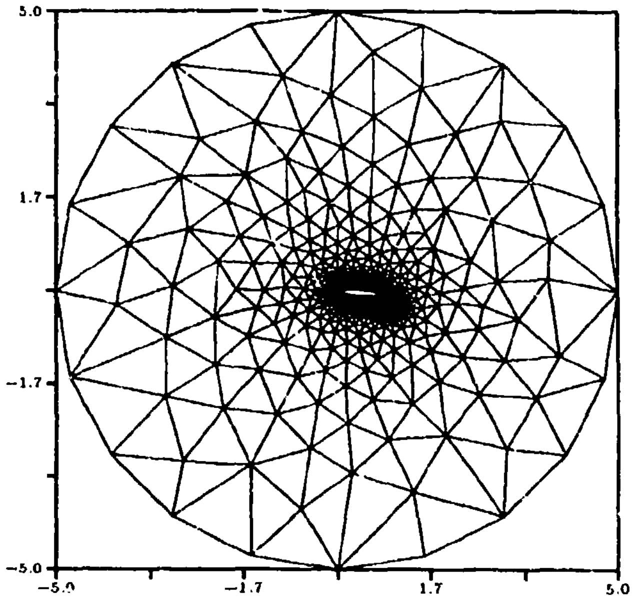


Fig. 10: Grid around three-element aerofoil.

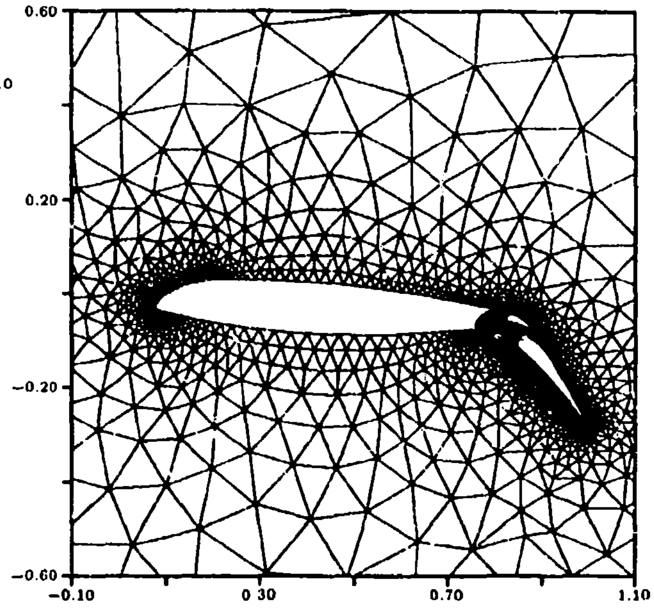


Fig. 11: Close-up of grid around three-element aerofoil.

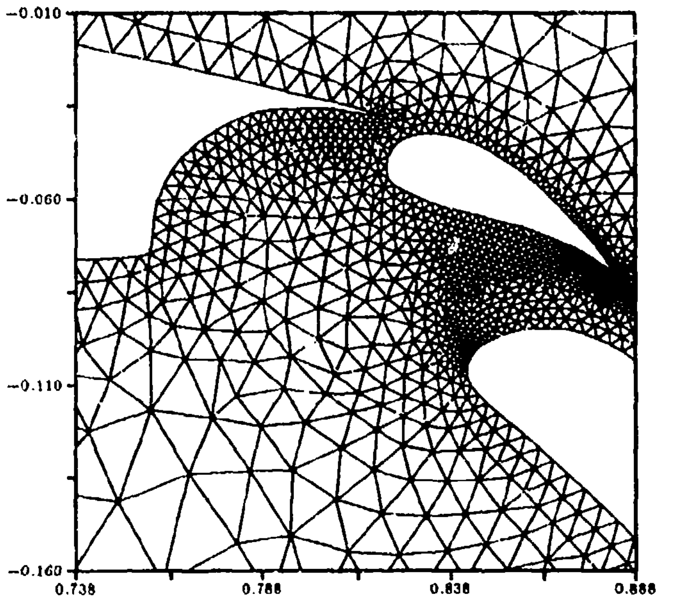


Fig. 12: Detail of grid around three-element aerofoil.

smoothly. Only very few triangles with maximum angles exceeding 90° can be found. If fronts are aligned to each other, the resulting point cloud is perfectly regular as between the main flap and the vane flap. The gradual increase in node spacing between the main aerofoil and the main flap is due to the automatic insertion of additional background nodes which are not present in the foreground grid. Overall, the cell surface varies very smoothly with a factor of about 100 000 from the smallest cells at the trailing edge of the vane flap to the largest cells at the outer boundary.
The only user input for the case were the 328 boundary nodes and a statement requiring no connection between the main aerofoil and the main flap.

5 CONCLUSIONS

A frontal mechanism for the creation of the interior nodes of a Delaunay triangulation has been developed. The method combines the high node distribution quality of the Advancing Front method with the optimal connectivity of the Delaunay triangulation. Precise control of node spacing is achieved by the use of the initial triangulation of the boundary nodes as a background mesh with no additional effort of the user. The node generation does not require explicit tracking of the front and is independent of the order in which triangles are listed.
We are presently working on a generalization that can incorporate stretching to obtain a non-isotropic triangulation. All features of this concept readily extend to three dimensions where the optimal operation count and the simplicity of boundary control, front tracking and node construction of the method become even more important.

6 REFERENCES

1. J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, Adaptive Remeshing for Compressible Flow Computations, Journal of Computational Physics 72, 1987.

2. D.G. Holmes and D.D. Snyder, The Generation of Unstructured Triangular Meshes Using Delaunay Triangulation, Proceedings of the Second Conference on Grid Generation in Computational Fluid Dynamics, Pineridge Press, Swansea, 1988.

3. B. Delaunay, Sur la sphère vide, Bull. Acad. Science USSR VII: Class. Sci. Mat. Nat. 793-800, 1934.

4. J. Peraire, K. Morgan, J. Piero and J. Bonet, Unstructured Mesh Methods for Computational Fluid Dynamics, Part 4, Data Structures, Von Karman Institute Lecture Series 1990-06 in Numerical Grid Generation.

5. D.T. Lee and B.J. Schachter, Two Algorithms for Constructing a Delaunay Triangulation, Int. Journal of Comp. and Inform. Sciences, Vol. 9, No. 3, 1980.

6. M.D. Rees, Numerical solution of the heat equation on triangular grids, Computing Laboratory, University of Oxford, Numerical Analysis Report 88/2, 1988.

7. T. Barth, On Unstructured Grids and Solvers, Von Karman Institute Lecture Series 1990-03 in Computational Fluid Dynamics.

8. S. Rippa, Minimal Roughness Property of the Delaunay Triangulation, PhD thesis, Tel-Aviv University, 1989.

9. T. Baker, Element Quality in Tetrahedral Meshes, presented at the 7th International Conference on Finite Element Methods in Flow Problems, Huntsville, Alabama, April 3-7 1989.

10. A. Bowyer, Computing Dirichlet Tessellation, The Computer Journal, Vol. 24, No. 2, 1981

11. N.P. Weatherill, Grid Generation, Part3, The Delaunay Triangulation, Von Karman Institute Lecture Series 1990-06 in Numerical Grid Generation.

12. J.-D. Müller, Frontal Node Generation for Delaunay Triangulations, Technical Note, Von Karman Institute, in preparation.

REPORT DOCUMENTATION PAGE

1. Recipient's Reference	2. Originator's Reference	3. Further Reference	4. Security Classification of Document
	AGARD-R-787	ISBN 92-835-0671-5	UNCLASSIFIED

5. Originator	Advisory Group for Aerospace Research and Development North Atlantic Treaty Organization 7 Rue Ancelle, 92200 Neuilly sur Seine, France
6. Title	SPECIAL COURSE ON UNSTRUCTURED GRID METHODS FOR ADVECTION DOMINATED FLOWS
7. Presented at	an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 2nd—6th March 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

8. Author(s)/Editor(s)	9. Date
Various	May 1992
10. Author's/Editor's Address	**11. Pages**
Various	364

12. Distribution Statement	This document is distributed in accordance with AGARD policies and regulations, which are outlined on the back covers of all AGARD publications.

13. Keywords/Descriptors

Finite element analysis	Aerodynamics
Grids - coordinates	Advection
Hypersonic flow	Numerical analysis
Transonic flow	Finite difference theory
Flow visualization	Reentry
Compressible flow	Reynolds number
Incompressible flow	

14. Abstract

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Unstructured Grid Methods for Advection Dominated Flows" have been assembled in this report. The objective of this course was to provide state-of-the-art information, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Specifically, topics and methods covered include:

— Least Squares Galerkin and Streamline Diffusion Finite Element Methods
— Finite Volume Methods and Higher Order Polynomial Reconstruction
— Essentially Non Oscillatory Schemes for Unstructured Grids
— Multidimensional Upwind Schemes on Triangles and Tetrahedra
— Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle
— Turbulence Modelling on Unstructured Grids
— Error Estimato [illegible] Solution Adaptivity
— Parallel Computing on Unstructured Grid
— Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, which includes incompressible free surface problems, transonic aerodynamics, and hypersonic reentry flows.

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented as an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 18th—22nd May 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

AGARD Report 787
Advisory Group for Aerospace Research and Development, NATO
SPECIAL COURSE ON UNSTRUCTURED GRID METHODS FOR ADVECTION DOMINATED FLOWS
Published May 1992
364 pages

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Unstructured Grid Methods for Advection Dominated Flows" have been assembled in this report. The objective of this course was to provide state-of-the-art information, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Specifically, topics and methods covered include:

P.T.O.

AGARD-R-787

Finite element analysis
Grids - coordinates
Hypersonic flow
Transonic flow
Flow visualization
Compressible flow
Incompressible flow
Aerodynamics
Advection
Numerical analysis
Finite difference theory
Reentry
Reynolds number

AGARD Report 787
Advisory Group for Aerospace Research and Development, NATO
SPECIAL COURSE ON UNSTRUCTURED GRID METHODS FOR ADVECTION DOMINATED FLOWS
Published May 1992
364 pages

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Unstructured Grid Methods for Advection Dominated Flows" have been assembled in this report. The objective of this course was to provide state-of-the-art information, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Specifically, topics and methods covered include:

P.T.O.

AGARD-R-787

Finite element analysis
Grids - coordinates
Hypersonic flow
Transonic flow
Flow visualization
Compressible flow
Incompressible flow
Aerodynamics
Advection
Numerical analysis
Finite difference theory
Reentry
Reynolds number

AGARD Report 787
Advisory Group for Aerospace Research and Development, NATO
SPECIAL COURSE ON UNSTRUCTURED GRID METHODS FOR ADVECTION DOMINATED FLOWS
Published May 1992
364 pages

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Unstructured Grid Methods for Advection Dominated Flows" have been assembled in this report. The objective of this course was to provide state-of-the-art information, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Specifically, topics and methods covered include:

P.T.O.

AGARD-R-787

Finite element analysis
Grids - coordinates
Hypersonic flow
Transonic flow
Flow visualization
Compressible flow
Incompressible flow
Aerodynamics
Advection
Numerical analysis
Finite difference theory
Reentry
Reynolds number

AGARD Report 787
Advisory Group for Aerospace Research and Development, NATO
SPECIAL COURSE ON UNSTRUCTURED GRID METHODS FOR ADVECTION DOMINATED FLOWS
Published May 1992
364 pages

Lecture notes for the AGARD Fluid Dynamics Panel (FDP) Special Course on "Unstructured Grid Methods for Advection Dominated Flows" have been assembled in this report. The objective of this course was to provide state-of-the-art information, as well as recent developments in unstructured grid methods, suitable for the computation of high Reynolds number compressible and incompressible flows, and other related subjects. Specifically, topics and methods covered include:

P.T.O.

AGARD-R-787

Finite element analysis
Grids - coordinates
Hypersonic flow
Transonic flow
Flow visualization
Compressible flow
Incompressible flow
Aerodynamics
Advection
Numerical analysis
Finite difference theory
Reentry
Reynolds number

- Least Squares Galerkin and Streamline Diffusion Finite Element Methods
- Finite Volume Methods and Higher Order Polynomial Reconstruction
- Essentially Non Oscillatory Schemes for Unstructured Grids
- Multidimensional Upwind Schemes on Triangles and Tetrahedra
- Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle
- Turbulence Modelling on Unstructured Grids
- Error Estimators and Solution Adaptivity
- Parallel Computing on Unstructured Grid
- Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, which includes incompressible free surface problems, transonic aerodynamics, and hypersonic reentry flows.

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented as an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 18th—22nd May 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

ISBN 92-835-0671-5

- Least Squares Galerkin and Streamline Diffusion Finite Element Methods
- Finite Volume Methods and Higher Order Polynomial Reconstruction
- Essentially Non Oscillatory Schemes for Unstructured Grids
- Multidimensional Upwind Schemes on Triangles and Tetrahedra
- Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle
- Turbulence Modelling on Unstructured Grids
- Error Estimators and Solution Adaptivity
- Parallel Computing on Unstructured Grid
- Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, which includes incompressible free surface problems, transonic aerodynamics, and hypersonic reentry flows.

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented as an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 18th—22nd May 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

ISBN 92-835-0671-5

- Least Squares Galerkin and Streamline Diffusion Finite Element Methods
- Finite Volume Methods and Higher Order Polynomial Reconstruction
- Essentially Non Oscillatory Schemes for Unstructured Grids
- Multidimensional Upwind Schemes on Triangles and Tetrahedra
- Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle
- Turbulence Modelling on Unstructured Grids
- Error Estimators and Solution Adaptivity
- Parallel Computing on Unstructured Grid
- Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, which includes incompressible free surface problems, transonic aerodynamics, and hypersonic reentry flows.

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented as an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 18th—22nd May 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

ISBN 92-835-0671-5

- Least Squares Galerkin and Streamline Diffusion Finite Element Methods
- Finite Volume Methods and Higher Order Polynomial Reconstruction
- Essentially Non Oscillatory Schemes for Unstructured Grids
- Multidimensional Upwind Schemes on Triangles and Tetrahedra
- Grid Generation Methods for Unstructured Grids Using the Frontal Method and Delaunay Principle
- Turbulence Modelling on Unstructured Grids
- Error Estimators and Solution Adaptivity
- Parallel Computing on Unstructured Grid
- Post Processing Unstructured Grid Data Bases for Flow Visualization Analysis

A wide range of applications is presented, which includes incompressible free surface problems, transonic aerodynamics, and hypersonic reentry flows.

The material assembled in this report was prepared and presented under the combined sponsorship of the AGARD Fluid Dynamics Panel, the Consultant and Exchange Programme of AGARD, and the von Kármán Institute (VKI) for Fluid Dynamics. It was presented as an AGARD-FDP-VKI Special Course at the VKI, Rhode-Saint-Genèse, Belgium, 18th—22nd May 1992 and at the NASA Ames Research Center, Moffett Field, California, United States, 28th September—2nd October 1992.

ISBN 92-835-0671-5

SPS

Imprimé par Specialised Printing Services Limited
40 Chigwell Lane, Loughton, Essex IG10 3TZ

AGARD

NATO OTAN

7 RUE ANCELLE · 92200 NEUILLY-SUR-SEINE

FRANCE

Telephone (1)47.38.57.00 · Telex 610 176
Telefax (1)47.38.57.99

DISTRIBUTION OF UNCLASSIFIED AGARD PUBLICATIONS

AGARD does NOT hold stocks of AGARD publications at the above address for general distribution. Initial distribution of AGARD publications is made to AGARD Member Nations through the following National Distribution Centres. Further copies are sometimes available from these Centres (except in the United States), but if not may be purchased in Microfiche or Photocopy form from the Sales Agencies listed below.

NATIONAL DISTRIBUTION CENTRES

BELGIUM
Coordonnateur AGARD-
Etat-Major de la
Quartier Reine El
Rue d'Evere, 114

CANADA
Director Scientific
Dept of National
Ottawa, Ontario K

DENMARK
Danish Defence R
Ved Idraetsparken
2100 Copenhagen

FRANCE
O.N.E.R.A. (Direct
29 Avenue de la Di
92322 Châtillon C

GERMANY
Fachinformationszc
Karlsruhe
D-7514 Eggenstein-

GREECE
Hellenic Air Force
Air War College
Scientific and Technical Library
Dekelia Air Force Base
Dekelia, Athens TGA 1010

ICELAND
Director of Aviation
c/o Flugrad
Reykjavik

ITALY
Aeronautica Militare
Ufficio del Delegato Nazionale all'AGARD
Aeroporto Pratica di Mare
00040 Pomezia (Roma)

NASA
National Aeronautics and Space Administration
Washington, D.C. 20546

SPECIAL FOURTH CLASS MAIL BOOK

Postage and Fees Paid
National Aeronautics and Space Administration
NASA-451

Official Business
Penalty for Private Use $300

U.S.MAIL

```
L2 001 AGARDR787 920716S002672D
DEPT OF DEFENSE
DEFENSE TECHNICAL INFORMATION CENTER
ATTN : DTIC-OCP/JOYCE CHIRAS
CAMERON STATION BLDG 5
ALEXANDRIA VA 223046145
```

28008 Madrid

TURKEY
Milli Savunma Başkanlığı (MSB)
ARGE Daire Başkanlığı (ARGE)
Ankara

UNITED KINGDOM
Defence Research Information Centre
Kentigern House
65 Brown Street
Glasgow G2 8EX

UNITED STATES
National Aeronautics and Space Administration (NASA)
Langley Research Center
M/S 180
Hampton, Virginia 23665

THE UNITED STATES NATIONAL DISTRIBUTION CENTRE (NASA) DOES NOT HOLD STOCKS OF AGARD PUBLICATIONS, AND APPLICATIONS FOR COPIES SHOULD BE MADE DIRECT TO THE NATIONAL TECHNICAL INFORMATION SERVICE (NTIS) AT THE ADDRESS BELOW.

SALES AGENCIES

National Technical Information Service (NTIS)	ESA/Information Retrieval Service	The British Library
5285 Port Royal Road	European Space Agency	Document Supply Centre
Springfield, Virginia 22161	10, rue Mario Nikis	Boston Spa, Wetherby
United States	75015 Paris	West Yorkshire LS23 7BQ
	France	United Kingdom

Requests for microfiches or photocopies of AGARD documents (including requests to NTIS) should include the word 'AGARD' and the AGARD serial number (for example AGARD-AG-315). Collateral information such as title and publication date is desirable. Note that AGARD Reports and Advisory Reports should be specified as AGARD-R-nnn and AGARD-AR-nnn, respectively. Full bibliographical references and abstracts of AGARD publications are given in the following journals:

Scientific and Technical Aerospace Reports (STAR)
published by NASA Scientific and Technical Information Division
NASA Headquarters (NTT)
Washington D.C. 20546
United States

Government Reports Announcements and Index (GRA&I)
published by the National Technical Information Service
Springfield
Virginia 22161
United States
(also available online in the NTIS Bibliographic Database or on CD-ROM)

SPS

Printed by Specialised Printing Services Limited
40 Chigwell Lane, Loughton, Essex IG10 3TZ